Graph Algorithmic Approaches for Structure Inferences in Bioinformatics(Dissertation_全文)

AUTHOR(S):

Tamura, Takeyuki

Graph Algorithmic Approaches for Structure Inferences in Bioinformatics

Takeyuki Tamura

Graph Algorithmic Approaches for Structure Inferences in Bioinformatics

Takeyuki Tamura

Graduate School of Informatics
Kyoto University
JAPAN

January 2006

Preface

This doctoral thesis is written on graph algorithmic approaches for structure inferences in bioinformatics. Since the double helix structure of DNA was first published by J. Watson and F. Crick in 1953, molecular biology that is the study of biology at a molecular level has been greatly developed. However, the drastical increase of gotten data caused that computer calculations became essential to inferring their biological meanings. Especially, the human genome that was completed to map down to the base pair level by the Human Genome Project (HGP) in 2003 consists of 3 billion base pairs. It is no doubt that theoretical construction of robust algorithms is necessary for these computer calculations with vast data. Since graphs are data structures to represent whether there is a specified relation between two objects or not, using graphs is one of the most practical modeling methods in order to infer biological meanings from gotten data. Then, effective graph algorithms play great roles in inferring various structures in bioinformatics.

The research after decoding of genomes is called post-genome. However, it is still important to sequencing new genomes. In order to research biological meanings of base-pairs, objects of sequencing are needed to extend to specified structures. Our first research is to infer probe orders with using PQ-trees as follows: In STS-based mapping, it is necessary to obtain the correct order of probes in a DNA sequence from a given set of fragments or an equivalently a hybridization matrix A. It is well-known that the problem is formulated as the combinatorial problem of obtaining a permutation of A's columns so that the resulting matrix has a consecutive-one property. If the data (the hybridization matrix) is error free and includes enough information, then the above column order uniquely determines the correct order of the probes. Unfortunately this does not hold if the data include errors, and this has been a popular research target in computational biology. Even if there is no error, ambiguities in the probe order may still remain. This in fact happens because of the lack of some information regarding the data, but almost no further investigation has previously been made. In this research, we define a measure of such imperfectness of the data as the minimum amount of the additional fragments that are needed to uniquely fix the probe order. Polynomial-time algorithms to compute such additional fragments of the minimum cost are presented. A computer simulation using genes of human chromosome 20 is also noted.

In the research mentioned above, the measure of the imperfectness is the number

of the additional fragments that are needed to uniquely fix the probe order. However, there is a possibility that better measures exist. In our second research, we define another measure of imperfectness and construct polynomial time algorithms that calculate minimum fragment sets. The second measure of imperfectness is the total length of the additional fragments that are needed to uniquely fix the probe order. We also mention cases that additional fragments are restricted to sub-fragments of existed fragments, and that the lengths of additional fragments are limited to 2.

Mendel's law, that was the trigger for the birth of genetics, was published in 1865. Needless to say, every individual has two parents in this law. In the third research, we infer this data structure that we call a pedigree graph. If every individual has only one parent, the data structure becomes an extended phylogenetic tree. A phylogenetic tree is a tree data structure which shows evolutionary interrelationships among species which are believed to have a common ancestor. On the other hand, pedigrees can be regarded as more informed phylogenetic trees. Phylogenetic trees can be represented by directed trees, whereas pedigrees are represented by directed acyclic graphs whose indegrees are at most two. In this research, we deal with a problem of enumerating all pedigrees which satisfy a given matrix of genetic distances between all pairs of n nodes. We show that the number of output pedigrees may be exponential, but they can be represented by one directed graph with n nodes. Moreover, $O(n^3)$ time algorithm which solves the problem is also given.

Although the Human Genome Project completed to decode human genome that consists of 3 billion base pairs, the biological meanings of the sequences are not revealed enough. We believe that our inferring algorithms would be of some help for revealing the biological meanings of genomes. And, we also hope that every post-genome research in the world would be supported by efficient algorithms that are constructed with robust theory.

January, 2006
Takeyuki Tamura

Acknowledgments

I would like to express my sincere appreciation to Professor Kazuo Iwama. Seven years have passed since I entered his laboratory. He gave me much opportunity for studying algorithms of various fields. Especially, he suggested me starting to research bioinformatics. I could hear and participate in many educational and leading discussions by him. The weekly meeting of his laboratory brought me up steadily. His precious words would have a favorable influence on my future activities. The tour of Hongkong for ISAAC2004 was memorable.

I would like to express my sincere appreciation to Professor Hiro Ito. He gave me opportunity for starting to study graph theory and graph algorithms. The seminar on graph theory led by him gave me much progress in these fields. Moreover, regular discussion with him played one of the greatest role in my research activities. Since he respected my autonomy, I was encouraged to find new themes and problems in bioinformatics. The tour of France for my first presentation at international conferences was very enjoyable.

I am deeply grateful to Professor Shuichi Miyazaki. He was indispensable to activities of the laboratory. I am also grateful to Professor Takashi Horiyama for supporting my research activities with his great contribution to Iwama laboratory.

My gratitude goes to Professor Tatsuya Akutsu. He advised us about bioinformatics. He introduced some biologists to me by e-mails. My gratitude also goes to Daiji Fukagawa, who was a member of Iwama laboratory and belongs to Akutsu laboratory now. I would like to thank Daisuke Tsuchida. My research on bioinformatics was started by reading his thesis.

I would like to thank all the members of Iwama laboratory for giving me helpful advice and encouragements for my research work. I would like to thank all of my friends for having a meaningful and enjoyable time together.

Finally, I would like to express my supreme gratitude to my family, for encouraging and supporting me at all times.

Contents

List of Figures

Chapter 1

Introduction

It is no doubt that theoretical construction of robust algorithms is necessary for computer calculations with vast data such as the human genome. Since graphs are data structures to represent whether there is a specified relation between two objects or not, using graphs is one of the most practical modeling methods in order to infer biological meanings from gotten data. Then, effective graph algorithms play great roles in inferring various structures in bioinformatics.

Our first research is to infer probe orders with using PQ-trees. A PQ-tree T consists of P-nodes denoted by circles, Q-nodes denoted by rectangles, and leaf-nodes. $P(T)$ denotes a set of permutations of leaf-nodes that is defined by the following rules: (i) Children of a P-node may be arbitrarily permuted. (ii) Children of a Q-node must be consecutive but may be arranged in reverse order.

A pedigree, in which every individual has two parents, is one of the most essential data structures in research of bioinformatics. However, it seems that there has been almost no research about inferring pedigrees from theoretical and algorithmic approaches. In this thesis, we infer this data structure, that we call a pedigree graph. If every individual has only one parent, the data structure becomes an extended phylogenetic tree. A phylogenetic tree is a tree data structure which shows evolutionary interrelationships among species which are believed to have a common ancestor. On the other hand, pedigrees can be regarded as more informed phylogenetic trees. Phylogenetic trees can be represented by directed trees, whereas pedigrees are represented

by directed acyclic graphs whose indegrees are at most two. In this research, we deal with a problem of enumerating all pedigrees which satisfy a given matrix of genetic distances between all pairs of n nodes. We show that the number of output pedigrees may be exponential, but they can be represented by one directed graph with n nodes. Moreover, $O(n^3)$ time algorithm which solves the problem is also given.

The overview of this thesis is as follows: In Chapter 2, we define a measure of imperfectness of PQ-trees as the minimum amount of the additional fragments that are needed to uniquely fix the leaf-node order. Polynomial-time algorithms to compute such additional fragments of the minimum cost are presented. A computer simulation using genes of human chromosome 20 is also noted.

STS-based mapping is one of the most popular techniques for physical mapping of DNA sequences. In this procedure, a DNA sequence S is cloned into many copies and these are then cut into smaller, overlapped subsequences called *fragments*. An STS (sequence-tagged site), also called a *probe*, is used as a marker; each probe is supposed to appear at a unique position in the entire DNA sequence S. We are given a *hybridization matrix*, an *H-matrix* in short, $A = (a_{ij})$ such that $a_{ij} = 1$ if probe p_j exists in fragment f_i and $a_{ij} = 0$ otherwise. Our goal is to compute the order of probes $P=\{p_1, \ldots, p_n\}$ in the original DNA sequence S from the given H-matrix A. It is well-known that this can be formulated as the following combinatorial problem: Given an H-matrix, obtain a permutation of the columns so that the resulting matrix has the so-called *consecutive-one property*, i.e., all 1s are consecutive in each row of the matrix.

We consider the problem that for a given H-matrix, obtain the minimum amount of additional fragments such that there is only one order of columns for the augmented H-matrix to have the consecutive-one property. There is an issue that should be taken into consideration, that is, the minimum amount of fragments differs according to the order of the probes to be selected as the unique one among possible different orders.

Our main result is to provide polynomial-time algorithms that compute (1) for a given H-matrix having a consecutive-one property, the minimum number of additional fragments that are sufficient to fix the probe order to the current order (i.e., the order of the columns in the given H-matrix), (2) for a given H-matrix not necessarily having a consecutive-one property, the minimum number of additional fragments sufficient to

uniquely fix the probe order (but the order itself may be arbitrary) so that the augmented H-matrix has a consecutive-one property. We also note a computer simulation using genes of human chromosome 20.

In Chapter 3, we define another measure of imperfectness and construct polynomial time algorithms that calculate minimum fragment sets. The second measure of imperfectness is the total length of the additional fragments that are needed to uniquely fix the probe order. This measure sometimes conflicts to the first measure, that is the minimum number of additional fragments. Moreover, the idea of edges cannot be used in this measure any longer. We also mention cases that additional fragments are restricted to sub-fragments of existed fragments, and that the lengths of additional fragments are limited to 2.

In Chapter 4, we deal with a problem of enumerating all pedigrees which satisfy a given matrix of genetic distances between all pairs of n nodes. We show that the number of output pedigrees may be exponential, but they can be represented by one directed graph with n nodes. Moreover, $O(n^3)$ time algorithm which solves the problem is also given.

A phylogenetic tree is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor. Since a phylogenetic tree doesn't include cycles, it cannot represent recombinations, that is, there are no vertices whose indegrees are more than one. Vertices of phylogenetic trees represent species. However, there are a huge number of individuals in each vertex and they have blood relationships each other. If all individuals are represented by vertices, indegrees of vertices should be two. However, since the number of individuals is finite, there must be a node whose indegree is one or zero. Moreover, there is a possibility that the number of vertices which don't have in-edges is more than one. Then, in this chapter, we discuss problems to infer directed graphs whose indegrees are at most two (We call them pedigree graphs). When genetic distances between any two individuals are given, we discuss problems (i) to enumerate all pedigree graphs which satisfy a given genetic distances. (ii) to validate whether there is a contradiction or not between a given distances and a given pedigree graph.

Finally, Chapter 5 concludes this thesis.

Chapter 2

Imperfectness of Data for STS-Based Physical Mapping

2.1 Introduction

STS-based mapping is one of the most popular techniques for physical mapping of DNA sequences. In this procedure, a DNA sequence S is cloned into many copies and these are then cut into smaller, overlapped subsequences called *fragments*. An STS (sequence-tagged site), also called a *probe* [55], is used as a marker; each probe is supposed to appear at a unique position in the entire DNA sequence S. We are given a *hybridization matrix*, an *H-matrix* in short, $A = (a_{ij})$ such that $a_{ij} = 1$ if probe p_j exists in fragment f_i and $a_{ij} = 0$ otherwise. Our goal is to compute the order of probes $P=\{p_1, \ldots, p_n\}$ in the original DNA sequence S from the given H-matrix A. It is well-known that this can be formulated as the following combinatorial problem: Given an H-matrix, obtain a permutation of the columns so that the resulting matrix has the so-called *consecutive-one property*, i.e., all 1s are consecutive in each row of the matrix.

The problem can be solved in linear time by using the famous data structure called *PQ-trees* [13]. Unfortunately, there are several kinds of errors involved in experiments, which makes the data, H-matrices in our case, imperfect. Typical errors include the case that (i) an entry of the H-matrix changes from 0 to 1, and vice versa, and that (ii)

two fragments, which are not consecutive in the DNA Sequence, are put together into a "chimeric" fragment [5, 9, 19, 20, 40, 47]. In the presence of such noises, we can no longer use PQ-trees; the problem now becomes one of several optimization problems due to different assumptions of the noises. Not surprisingly, they are NP-hard in most cases [5, 10, 19, 20, 30, 33, 48].

Even if there are no such errors, there may still remain ambiguities in the probe order. See for example Fig. 2.1 (a), which illustrates an example of an H-matrix consisting of six fragments (rows) f_1 to f_6, and ten probes (columns) A to J. By exchanging columns, the matrix can be transformed into the matrix in Fig. 2.1 (b) which satisfies the consecutive-one property, (i.e., each row has a single block of consecutive ones). However, one can see that there are several other orders of the columns, say EGBFIADHCJ, which also achieve a consecutive-one property. Thus we cannot uniquely fix the order of probes from the requirement of the consecutive-one property in the case of this H-matrix, which is obviously due to the imperfectness of the data. There are few reports mentioning the existence of this fact, e.g., [37, 57], but no further investigation followed.

In this chapter, we propose a measure of such imperfectness in H-matrices. Recall that the imperfectness is due to the lack of information. For example, if we add two extra fragments to the H-matrix of Fig. 2.1 (a) as in Fig. 2.1 (c), then the order of probes is now uniquely determined as shown in Fig. 2.1 (d). Thus the amount of additional fragments needed to uniquely fix the probe order looks closely related to the degree of the imperfectness. It appears convenient to know this quantity for conducting STS-based physical mapping.

More formally we consider the problem that for a given H-matrix, obtain the minimum amount of additional fragments such that there is only one order of columns for the augmented H-matrix to have the consecutive-one property. There is an issue that should be taken into consideration, that is, the minimum amount of fragments differs according to the order of the probes to be selected as the unique one among possible different orders. For example, we needed two additional fragments in Fig. 2.1 (d), but three additional fragments are needed to fix the column order as BGEAIDHCFJ shown in Fig. 2.1 (e).

Our main result is to provide polynomial-time algorithms that compute (1) for a

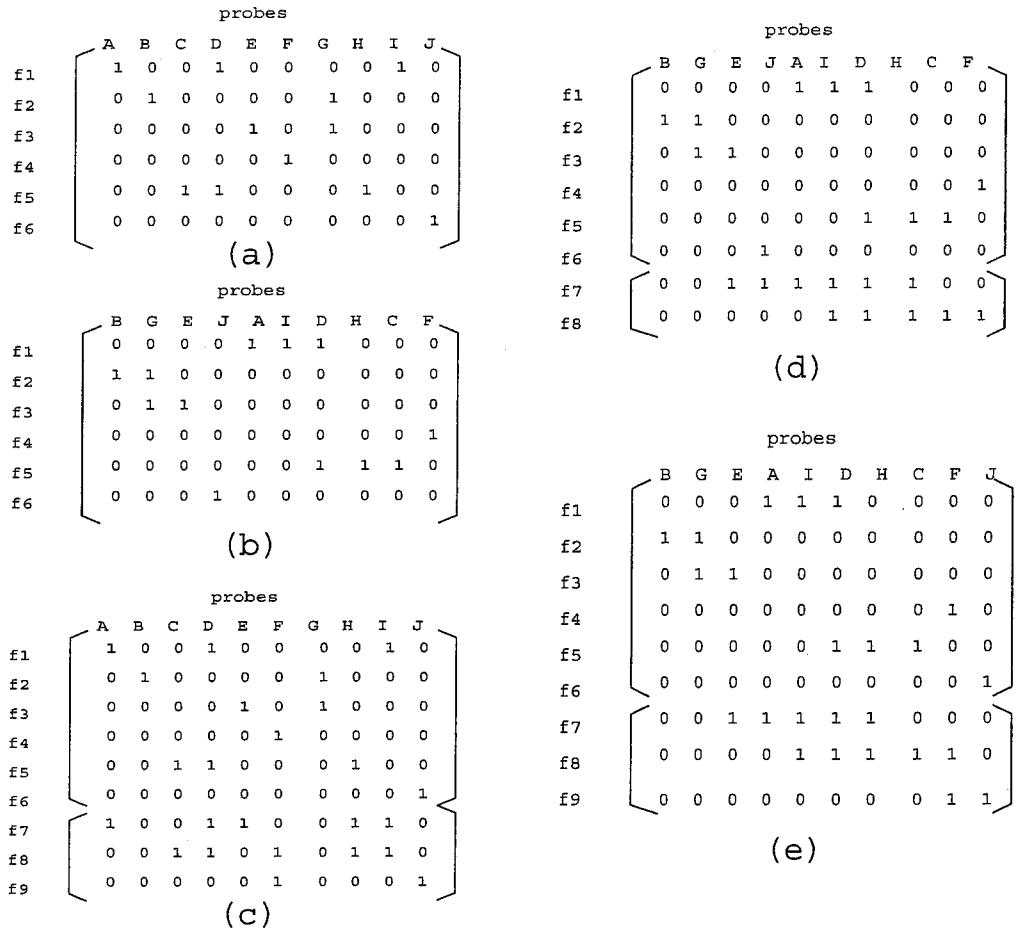


Figure 2.1: Permuting the (0,1)-matrix gotten by the experiment

given H-matrix having a consecutive-one property, the minimum number of additional fragments that are sufficient to fix the probe order to the current order (i.e., the order of the columns in the given H-matrix), (2) for a given H-matrix not necessarily having a consecutive-one property, the minimum number of additional fragments sufficient to uniquely fix the probe order (but the order itself may be arbitrary) so that the augmented H-matrix has a consecutive-one property. We also note a computer simulation using genes of human chromosome 20.

As mentioned, if data are perfect, then the problem can be solved in linear time by using PQ-trees [13]. Several possibilities of errors have been investigated including

obtaining a sub-matrix that has a consecutive-one property [29, 34], obtaining most-likely probe orders in the presence of a false position and false negative hybridization errors using a different data structure [20, 56], using the LP-relaxation for optimizing the most-likely probe order [40], and exploiting the fact that each probe occurs at a unique position by utilizing a more sophisticated way to handle errors such as chimeric fragments [5]. Also see [9, 10, 19, 47, 77] for related work including parallelization of the construction of PQ-trees [7].

There are various types of data that are used as inputs of problems including mapping by non-unique probes [4, 60], mapping by unique probes [5, 30, 40], mapping by unique end-probes [19], mapping by non-overlapping probes [20], mapping from restriction-fragment length data [23, 44], radiation-hybrid mapping [10, 68], optical mapping [46, 50, 62], and a maximum-likelihood approach for the sampling-without-replacement protocol [47]. There are many probabilistic analyses of various approaches [8, 49, 77, 78, 80] A wide variety of techniques have been suggested including greedy algorithms [60], simulated annealing [4, 5, 61, 74], linear programming [19, 20, 40], and semidefinite programming [17].

2.2 PQ-trees

PQ-trees are a convenient data structure for our problem. Fig. 2.2 shows an example of a PQ-tree. A PQ-tree T consists of *P-nodes* denoted by circles, *Q-nodes* denoted by rectangles, and *leaf-nodes*. $P(T)$ denotes a set of permutations of leaf-nodes that is defined by the following rules: (i) Children of a P-node may be arbitrarily permuted. (ii) Children of a Q-node must be consecutive but may be arranged in reverse order. For example, let T_0 be the PQ-tree in Fig. 2.2. Then $P(T_0)$=$\{BGEJAIDHCJF, EGBJIADHCF, \ldots\}$. Two PQ-trees T and T' are said to be equivalent if $P(T) = P(T')$.

There is a linear-time algorithm [13] that constructs a PQ-tree T from H-matrix A such that (i) T's leaf-nodes correspond to columns of A and (ii) A has a consecutive-one property if A's columns are rearranged into an order in $P(T)$. (If A cannot be rearranged into any matrix having a consecutive-one property, then the algorithm can detect it. If A is an H-matrix, this does not happen unless A includes errors.) Although

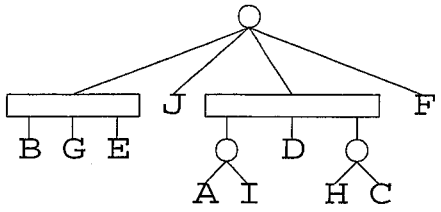


Figure 2.2: After adding fragment 5

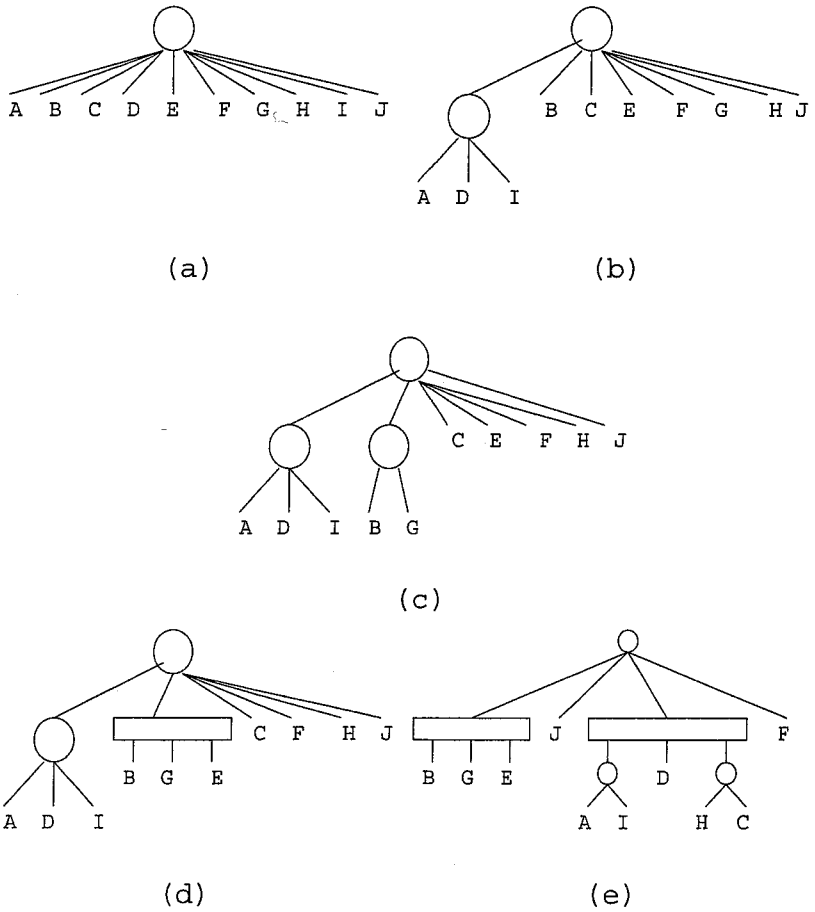


Figure 2.3: The process to make the input PQ-tree

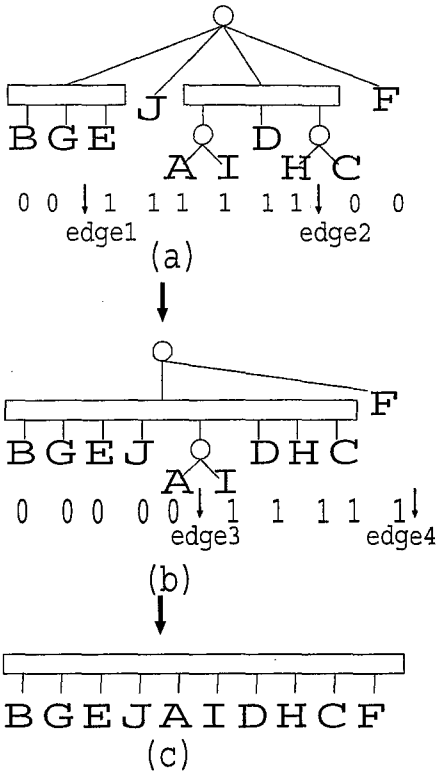


Figure 2.4: Making a 1Q-tree

details are omitted, the algorithm constructs a target PQ-tree by transforming PQ-trees step-by-step beginning with a PQ-tree of a single P-node. In each step, a row of the H-matrix is selected and the PQ-tree changes so that the constraint from that row is added by the templates shown in the appendix. For example, from the H-matrix in Fig 1(a), we can construct the associated PQ-tree as shown in Fig. 2.3 by selecting rows f_1 through f_6 in each step. Note that the final PQ-tree is the same as T_0 in Fig. 2.2 and $P(T_0)$ includes several different orders as mentioned before. For example, BGEJAIDHCF in $P(T_0)$ corresponds to the H-matrix in Fig. 2.3 (b) which has a consecutive-one property.

If we add two new rows (fragments) f_7 and f_8 as in Fig. 2.1 (c), then the PQ-tree is furthermore changed as in Fig. 2.4 and the final PQ-tree consists of a single Q-node. (Such a PQ-tree is called a *1Q-tree.*) This means that the probe order is uniquely fixed (without its reverse order) by adding two extra fragments, which is exactly what we

wanted to do. Thus our problem can be restated as follows.

Problem $FIX(T, \sigma, \sharp)$: For a given PQ-tree T (made from H-matrix by the algorithm of [13]) and a probe order (leaf order) σ, obtain a set of additional fragments of a minimum cost such that T will change into a 1Q-tree of leaf order σ.

If σ is not given then the problem is denoted by $FIX(T, -, \sharp)$ which requires that a set of additional fragments of a minimum cost be obtained to change T into *some* 1Q-tree. As the cost of a fragment set, we consider the size of the fragment set, i.e., the number of fragments.

2.3 Templates of PQ-trees

The method in [13] that makes a PQ-tree from an H-matrix is as follows:

begin

- **1:** Make a *universal tree* that has as many elements as columns. A universal tree has a single P-node for its root and leaves for every elements.
- **2:** Repeat the following procedures for every row of the matrix.
  - **2-1:** Put "Full" to leaves that are assigned 1. Put "Null" to leaves that are assigned 0.
  - **2-2:** Repeat the following procedures from leaves to the root.
    - **2-2-1:** Put nodes "Full", "Null" or "Partial" according to the following definitions.
    - **2-2-2:** Choose a template shown below.
      If any templates can't be applied, inputs are wrong and return an error, i.e., there is no feasible permutation.

end

Definitions of "Null", "Full" and "Partial" are as follows.

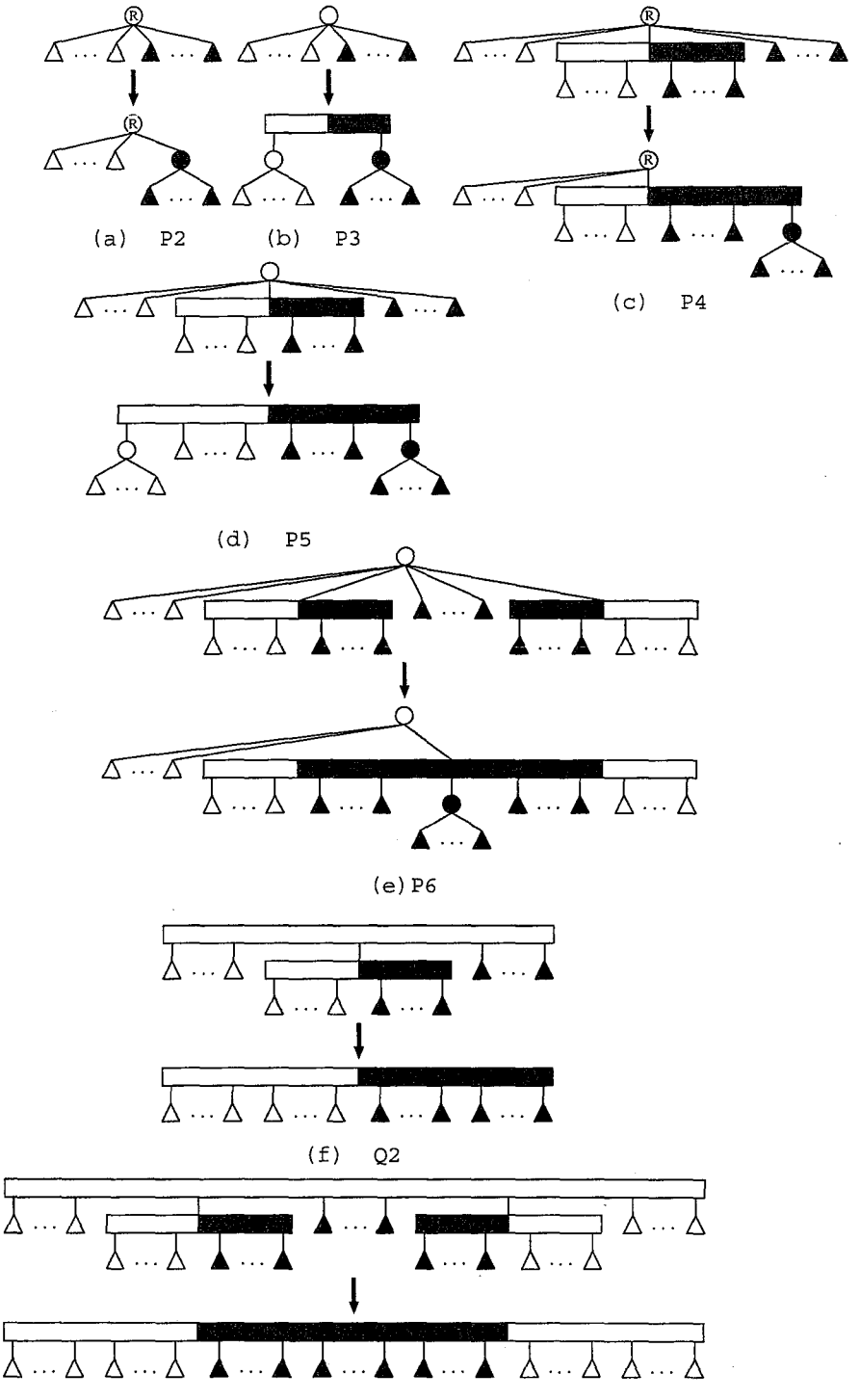


Figure 2.5: Templates for PQ-trees

NULL: There are no "Full" nodes in the subtree whose root is the current node.

FULL: The nodes in the subtree whose root is the current node are all "Full".

PARTIAL: The node is neither "Null" nor "Full".

Instructions for the templates are as follows. A hatched triangle means a Full node. A blank triangle means a Null node.

Template P0, P1 The templates for a P-node that has no partial child nodes. The template P0 is for a P-node whose child nodes are all "Null". When the template P0 is applied, the P-node is assigned "Null".

The template P1 is for a P-node whose child nodes are all "Full". When the template P1 is applied, the P-node is assigned "Full". In P0 and P1, the PQ-tree isn't transformed.

Template P2,P3 The templates for a P-node that has no partial child nodes. If there is no "Full" node outside of the subtree whose root is the current P-node, the template P2 is applied. The tree is transformed like in Fig. 2.5 (a).

If there is a "Full" node outside of this subtree, the template P3 is applied. In these two cases, The PQ-tree is transformed. The tree is transformed like in Fig. 2.5 (b).

Template P4,P5 The templates for a P-node that has only one "Partial" child node. If there are "Full" nodes outside of the subtree whose root is the current P-node, the template P4 is applied. The tree is transformed like in Fig. 2.5 (c).

If there is a "Full" node outside of this subtree, the template P5 is applied. The tree is transformed like in Fig. 2.5 (d).

Template P6 The template for a P-node that has two "partial" child nodes. The tree is transformed like in Fig. 2.5 (e).

Template Q0, Q1 The template for a Q-node that has no partial child node. The template Q0 is for a Q-node whose child nodes are all "Null". When the template Q0 is applied, the Q-node is assigned "Null".

The template Q1 is for a Q-node whose child nodes are all "Full". When the template Q1 is applied, the Q-node is assigned "Full". In Q0 and Q1, the PQ-tree isn't transformed.

Template Q2 The template for a Q-node that has only one partial child node. The tree is transformed like in Fig. 2.5 (f).

Template Q3 The template for a Q-node that has two partial child nodes. The tree is transformed like in Fig. 2.5 (g).

2.4 Minimizing the Number of Additional Fragments

In this section, we first discuss minimizing the number of additional fragments for $FIX(T, \sigma, \sharp)$ (i.e., the probe order is to be fixed to σ which is explicitly given) and then for $FIX(T, -, \sharp)$ (to be fixed to an arbitrary order).

2.4.1 $FIX(T, \sigma, \sharp)$

Suppose that the PQ-tree is given so that the leaves are arranged in the order $\sigma = p_1 p_2 \ldots p_n$ of length n. Then we consider $n+1$ different *positions*, denoted by $(-, p_1)$, (p_1, p_2), $\ldots$, (p_{n-1}, p_n), and $(p_n, -)$. Thus, a position means a "between" of two consecutive probes or the left (right) of p_1 (p_n). A position denoted by (p_i, p_{i+1}) is called an *inside* position, $(-, p_1)$ and $(p_n, -)$ an *outside* position. See Fig. 2.4 again. An additional fragment should have a consecutive sequence of probes, EJAIDH for example for the first added fragment in Fig. 2.4, which can be designated by giving two positions, its *left end-position* and *right end-position* ((G,E) and (H,C)) in the example). We sometimes say that a fragment is *terminated* by its (left and/or right) end-positions.

In Fig. 2.4 (a), we selected two positions (G,E) and (H,C) to terminate the first additional fragment. As can be seen later, this selection of (G,E) and (H,C) contributes to efficiently converting the PQ-tree into the final 1Q-tree. Thus among all positions, there are some "important" positions for our purpose. We call such positions "edges," since using these important positions as edges of additional fragments plays a major role in minimizing the number of additional fragments. *Edges* are divided into three

types and can be defined as follows: A position (x, y) is called (i) an *Inside-P-type edge* if probes x and y are children of a single P-node, (ii) an *Outside-P-type edge* if probe x (or y) is $-$ and it is a child of the root P-node, (iii) a *Q-type edge* if both x and y belong to a single Q-node which is not a root Q-node and which includes only leaf-nodes. In Fig. 2.4 (a) for example, (A,I) is Inside-P-type, (F,-) is Outside-P-type and (G,E) is Q-type. It should be noted that if we appropriately select two edges to terminate an additional fragment, like (G,E) and (H,C) in Fig. 2.4 (a) then those two edges "disappear" in the transformed PQ-tree ((G,E) or any other Q-type edge for the Q-node BGE). Thus the key point is how to select such appropriate edges for additional fragments.

By definition, (B,G) is also a Q-type edge for the same Q-node. However, we only need one Q-type edge for a Q-node for the fixing operation. In this case, {(B,G), (A,I), (H,C), (F,-)} or {(G,E), (A,I), (H,C), (F,-)} is the edge set that should be used by the fixing operation. Then, we say the number of edges that should be used by the fixing operation is four in Fig. 2.4 (a).

Lemma 2.1 *A PQ-tree includes no edge if and only if it is a 1Q-tree.*

Proof. If a PQ-tree has two internal nodes, there is at least one edge by the definition. If a PQ-tree has only one internal node and if it is a P-node, it includes at least one P-type edge from the definition. □

Lemma 2.2 *For any solution of $FIX(T, \sigma, \sharp)$, every edge must be selected at least once to terminate additional fragments.*

Proof.

- **Inside-P-type:**

  Assume that an Inside-P-type edge is not used to terminate additional fragments in a fixing operation and the 1Q-tree is gotten. Since the probe order is not changed in $FIX(T, \sigma, \sharp)$, the Inside-P-type edge never disappear. Then, this 1Q-tree has an edge. It is a contradiction. Thus, Inside-P-type edges must be selected to terminate additional fragment in fixing operations.

- **Q-type:**

  Assume that p_i and p_{i+1} are leaves and that the parent node of p_i and p_{i+1} is the same Q-node. Since we can assume that the given PQ-tree is not a 1Q-tree, the position (p_i, p_{i+1}) is the Q-type edge from the definition. At this point, the PQ-tree has at least 2 internal nodes. If any (p_i, p_{i+1}) is selected to terminate additional fragments in a fixing operation, the number of internal nodes does not become 1. Thus, Q-type edges must be selected to terminate additional fragment in fixing operations.

- **Outside-P-type:**

  Suppose that the root of the given PQ-tree is a P-node and p_n (p_1) is the rightmost (leftmost) probe. Since the proof for p_n can be applied for p_1, we discuss only about p_n. If p_n is a child node of the root P-node, the position $(p_n, -)$ is an Outside-P-type edge from the definition. Suppose that the position $(p_n, -)$ is not selected to terminate additional fragments in a fixing operation and the 1Q-tree is gotten. Although the probe order of the 1Q-tree must be $p_1 p_2 \ldots p_n$ in $FIX(T, \sigma, \sharp)$, there is a possibility that the probe order is $p_n p_1 p_2 \ldots p_{n-1}$. Then, it is a contradiction. Thus, Outside-P-type edges must be selected to terminate additional fragment in fixing operations.

□

In Fig. 2.4, the first additional fragment is terminated by edges (G,E) and (H,C). After adding this fragment, edges 1 and 2 disappear. However, we cannot say that every edge always disappears when a fragment terminated by the edge is added. For example, if the first additional fragment is terminated by (A,I) and (H,C), two Inside-P-type edges seem to disappear. However, because (A,I), (I,D), (D,H) and (H,C) become Q-type edges, the number of edges which are disappeared by this additional fragment is actually only one. In Fig. 2.4 (a), edges 1 and 2 have another edge, edge 3, between them. In fact, both edges always disappear in such a case as shown in the following lemma.

Lemma 2.3 *Suppose that a PQ-tree T_1 has two edges e_1 and e_2, and T_1 is transformed into T_2 by adding the fragment terminated by e_1 and e_2. Then (i) at least one of e_1 and e_2 disappears in T_2 and (ii) if there is another edge, say e_3, between e_1 and e_2, then both e_1 and e_2 disappear in T_2 (iii) Furthermore no new edges are created.*

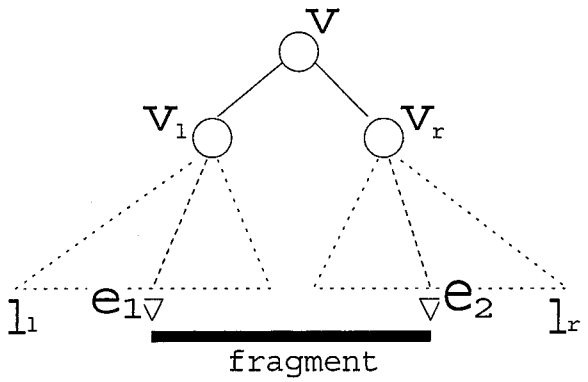


Figure 2.6: An additional fragment terminated by e_1 and e_2

Proof. Let v be the lowest common ancestor of e_1 and e_2. Let v_l be the internal node which is an ancestor of e_1 and a child of v. Let v_r be the internal node which is an ancestor of e_2 and a child of v. Let l_1 be the leftmost probe included in the subtree whose root is v_l. Let l_r be the rightmost probe included in the subtree whose root is v_r. (See Fig. 2.6)

By examining all templates for transformation of PQ-trees in each step defined in [13], property (iii) can be proved. Assume that a fragment terminated by e_1 and e_2 is added and only one of the two edges disappear. By examining all templates of [13], it can be proved that there are only the following two cases.

- When v is a P-node and there is not another edge except for e_1 and e_2 in the position set between l_l and l_r, (If there is another edge, two edges disappear.)
- When v is a Q-node and there is not another edge except for e_1 and e_2 in the positions included by the subtree whose root is v. (If there is another edge, two edges disappear.)

Hence, (i) and (ii) are shown. □

By using Lemma 2.3, we can remove two edges by adding one fragment, and thus we can show that the number of necessary additional fragments for a fixing operation is about a half of the number of edges. Note that, there must be at least three edges

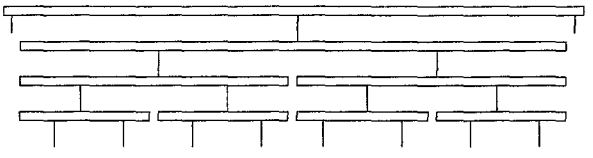


Figure 2.7: A PQ-tree in which $\frac{e}{2}+1$ additional fragments are necessary

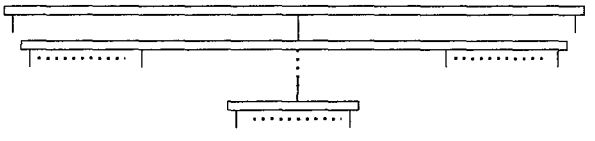


Figure 2.8: One step before the 1Q-tree

in order to apply Lemma 2.3. In fact, there exists a case that there are only two edges, and two fragments are needed. As the result of this, there exists a PQ-tree that has e edges and $\lceil\frac{e}{2}\rceil+1$ additional fragments are needed. The PQ-tree in Fig. 2.7 is an example of this; it becomes the PQ-tree of Fig. 2.8 after adding $\lceil\frac{e}{2}\rceil$ fragments terminated by two.

In other words, when the number of edges is even, there are two cases, i.e., the minimum numbers of additional fragments are $\frac{e}{2}$ and $\frac{e}{2}+1$. We can distinguish them by using a simple characterization, as in the following theorem.

Theorem 2.1 *Let e be the number of edges and n be the number of probes of $(T,\sigma,\sharp)$. The minimum number of additional fragments for $FIX(T,\sigma,\sharp)$ is shown as follows:*

1. *When e is odd: $\frac{e+1}{2}$.*
2. *When e is even:*

   2-1. When the root node is a Q-node and there is only one internal child node of the root: $\frac{e}{2}+1$.

   2-2. Otherwise: $\frac{e}{2}$.

Moreover, a fragment set with the minimum number of additional fragments for $FIX(T,\sigma,\sharp)$ can be found in $O(n^3)$ time.

For proving the theorem, we introduce the following lemma.

Lemma 2.4 *Consider a PQ-tree $(T, \sigma, \sharp)$ that includes at least three edges and does not satisfy the condition of 2-1 in Theorem 2.1. There exists a fragment satisfying the condition of Lemma 2.3 (ii) such that the resultant PQ-tree $(T', \sigma, \sharp)$ also does not satisfy the condition of 2-1 in Theorem 2.1 after adding the fragment.*

Proof of Theorem 2.1.

- When e is odd:

  From Lemma 2.3 (ii), two edges can be decreased by adding one fragment if $e \geq 3$. Hence, by iterating this process, only one edge remains after adding $\frac{e-1}{2}$ fragments. A PQ-tree including only one edge must satisfy all of the three conditions:

  - The root is a Q-node.
  - Every internal node has at most one internal child node.
  - The lowest internal node (the internal node which does not have an internal child node.) is a Q-node.

  It becomes a 1Q-tree by adding a fragment.

- When e is even:

  By using the same discussion with the odd case above, a PQ-tree including only two edges can be obtained by adding $\frac{e}{2} - 1$ fragments. From Lemma 2.4, if the original PQ-tree does not satisfy the condition of 2-1, then the resultant PQ-tree also does not. Hence, it is sufficient to consider the case where $e = 2$. It can be easily proved by examining all cases.

Because we consider only the given order of probes, there are $O(n^2)$ fragments. A transformation by each additional fragment can be done in $O(n)$ time.

□

2.4.2 $FIX(T, -, \sharp)$

The result of Theorem 2.1 can also be used to solve $FIX(T, -, \sharp)$. That is, $FIX(T, -, \sharp)$ can be solved by finding a leaf order σ in which the number of edges is the minimum. The following Lemma 2.5 shows how to find such σ. In the lemma, v and l mean the number of internal child nodes and the number of child probes, respectively, of the noted P-node.

Lemma 2.5 *Let a_1 be the number of Q-nodes that don't have internal child nodes. Let a_2 be the total number of $max\{|l| - |v| - 1, 0\}$ for all P-nodes that are not the root. Let a_3 be $max\{|l| - |v| + 1, 0\}$ if the root is a P-node, or 0 otherwise. The minimum number of edges for $FIX(T, -, \sharp)$ is $a_1 + a_2 + a_3$.*

It can be proved by definitions of the edges. In Fig. 2.2, the number of edges is four. If probe J is moved to the space between AIDHC and F, the number of edges becomes five.

Theorem 2.2 *A fragment set for $FIX(T, -, \sharp)$ can be found in $O(n^3)$ time, where n is the number of probes.*

Proof. It is clear from Lemma 2.3 and 2.5, and Theorem 2.1. □

2.5 Computer Experiments

We presented algorithms for finding the "minimum" set of fragments that fixes the order of probes. In this algorithm, the concept of edges introduced by us plays a major role. For practical use, it may be difficult to make additional fragments as we want. However, if fragments are concentrated to the part where edges exist densely, the probability that fragments that our algorithm wants are generated becomes high. In other words, the probability that edges disappear becomes high and fixing operations are accelerated.

For example, one concrete method is as follows: We generate some relatively long *sample fragments* at random. When the problem is $FIX(T, \sigma, \sharp)$, we should find the

densest fragment that includes more edges than any other sample fragment. When the problem is $FIX(T, -, \sharp)$, the numbers of edges included by sample fragments depend on the probe order. In this case, we adopt their maximum value for measuring the density. By cloning and cutting the densest fragment, sub-fragments of this fragment are obtained. The sub-fragments will be used as additional fragments. This method will accelerate fixing operations in comparison with a naive method that randomly generates additional fragments.

We validate this method by computer experiments. The thin broken line in Fig. 2.9 shows the result of the former method, and the thick solid line the result of the latter. The former method needs 7729 fragments (the average of 5 other fragment sets) to decrease the number of edges to $\frac{1}{10}$ of the initial state. However, the latter method needs only 4473 fragments. The ratio is 1.73.

Data Used in the Computer Experiment

Genes of human chromosome 20 are used as probes. We referred to data from HGREP (http://hgrep.ims.u-tokyo.ac.jp/); however, this site is not available any longer, though similar data are available from NCBI Human Genome Resources (http://www.ncbi.nlm.nih.gov/genome/guide/human/). In principle, 2500 base pairs from the beginning of the coding sequence of each gene are used as a probe. When more than 500 base pairs of probe j are included by the fragment i, (i, j) of the H-matrix is 1.

We compare the two methods mentioned by observing how the number of edges decreases. In principle, fragments are generated at random and Their lengths are normally distributed.

Let $N(\mu, \sigma, \sharp)$ be a normal distribution in which the mean is μ and the standard deviation is σ. In the computer experiment with the former method, fragments are randomly generated and their lengths are by $N(250000, 50000)$. In the latter method, sample fragments are generated at random and their lengths are by $N(1000000, 50000)$. Let f_1 be the densest fragment that includes at most e_1 edges. Let f_2 be the second densest fragment that includes at most e_2 edges. Additional fragments are sub-fragments of f_1 and the lengths are by $N(250000, 50000)$. When the number of edges in f_1 becomes less than or equal to e_2, new sample fragments are taken. We also take new sample fragments after generating 10 sub-fragments of the densest fragment.

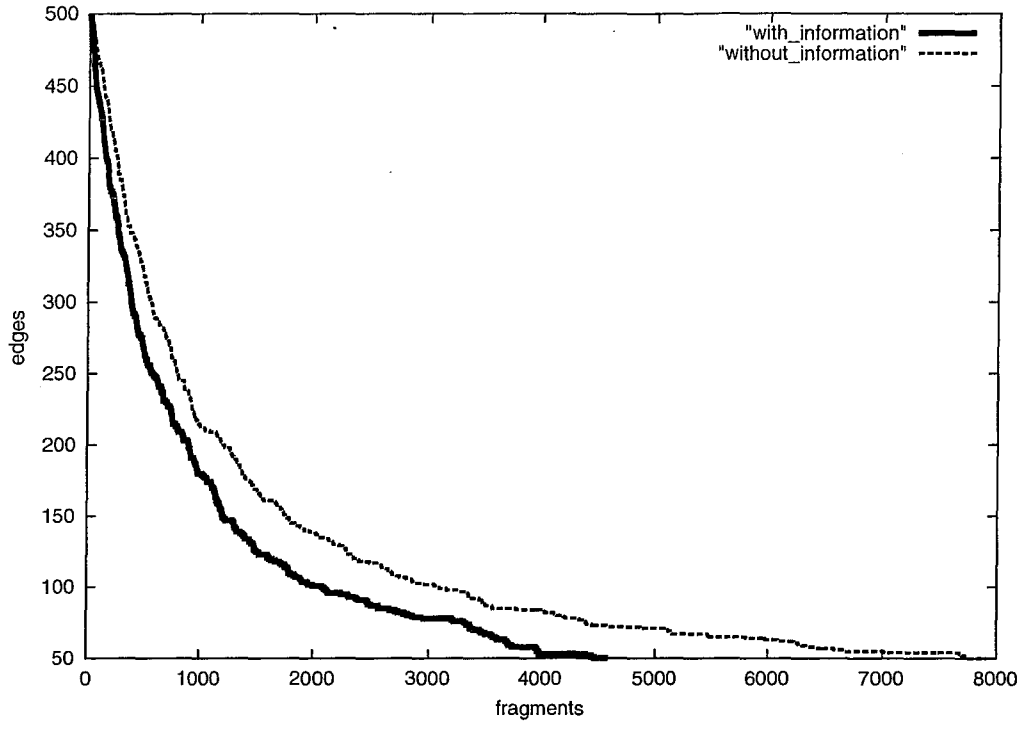


Figure 2.9: With information of edge-density or not

2.6 Concluding Remarks

For the problem of fixing the probe order of a given PQ-tree, we presented polynomial time algorithms that minimize the number of additional fragments. We solved not only the problem of fixing probes in a given order, but also the problem to how to determine the best order of the probes. By using an idea of "edges", we showed that the minimum number of additional fragments are $\lceil \frac{e}{2} \rceil$ or $\lceil \frac{e}{2} \rceil + 1$, where e is the number of edges.

For practical use, there is the possibility that H-matrices include errors. Then, a future work of ours will be to consider STS-based mapping with errors.

Chapter 3

Minimum Fragments for Deciding Probe Orders for DNA Strands

3.1 Introduction

In Chapter 2, we formulated problems $FIX(T, \sigma, \sharp)$ and $FIX(T, -, \sharp)$. These were defined as follows:

Problem $FIX(T, \sigma, \sharp)$: For a given PQ-tree T (made from H-matrix by the algorithm of [13]) and a probe order (leaf order) σ, obtain a set of additional fragments of a minimum cost such that T will change into a 1Q-tree of leaf order σ.

Problem $FIX(T, -, \sharp)$: If σ is not given then the problem is denoted by $FIX(T, -, \sharp)$ which requires that a set of additional fragments of a minimum cost be obtained to change T into *some* 1Q-tree.

The measure of the cost was the number of additional fragments in the previous chapter. On the other hand, we discuss about the total lengths of additional fragments in this chapter. Those two measures sometimes conflict. As a simple example, the PQ-tree in Fig. 3.1 needs one fragment of length five (i.e., the fragment including probes H, I, J, K and L) for the fixing operation. The same fixing operation is also possible by using two fragments of total length four as shown in Fig. 3.2.

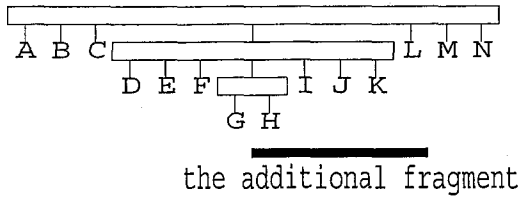


Figure 3.1: The number of fragments = 1
the total length of fragments = 5

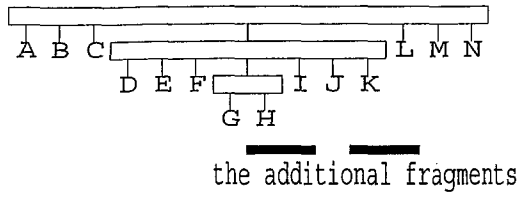


Figure 3.2: The number of fragments = 2
the total length of fragments = 4

Our main result of this chapter is to provide polynomial-time algorithms for following problems:

Problem $FIX(T, \sigma, \Sigma)$: For a given H-matrix having a consecutive-one property, the minimum total length of additional fragments that are sufficient to fix the probe order to the current order (i.e., the order of the columns in the given H-matrix),

Problem $FIX(T, -, \sharp)$: For a given H-matrix not necessarily having a consecutive-one property, the minimum total length of additional fragments sufficient to uniquely fix the probe order (but the order itself may be arbitrary) so that the augmented H-matrix has a consecutive-one property.

$FIX(T, \sigma, \Sigma)$ is mentioned in section 3.2.1. $FIX(T, -, \Sigma)$ is mentioned in section 3.2.2. The case in which additional fragments are limited to sub-fragments of existed fragments in $FIX(T, \sigma, \sharp)$ is mentioned in section 3.3. Cases in which lengths of additional fragments are limited to 2 in $FIX(T, \sigma, \sharp)$ and $FIX(T, \sigma, \Sigma)$ are mentioned in section 3.4.

3.2 Minimizing the Total Length of Additional Fragments

3.2.1 $FIX(T, \sigma, \Sigma)$

As shown in Fig.s 3.1 and 3.2, the smallness for the number of additional fragments and the shortness for the total length of additional fragments may conflict each other.

For a fixing operation, for every edge, there must be at least one fragment terminated by the edges. However, there is a case that we can shorten the total length by using a fragment which is terminated by two non-edge positions. The fragment which consists of KL shown in Fig. 3.2 is an example of this. As shown in this example, "edge" is a concept related to the number of fragments, and there is scarcely any relation between edges and the total lengths of fragments.

We propose an algorithm, which scans from the leaves to the root and is based on a dynamic programming, for this problem. We explain the basic ideas by using simple examples. Before the explanation, we introduce some notations as follows. A fragment *covers* position (i, j) if the fragment includes both i and j. A set F of fragments *covers* a set P of consecutive probes if for each neighbor probes $i, j \in P$, F has a fragment that covers position (i, j). If a set F of fragments doesn't cover a set P of consecutive probes, then there is at least one "cut" defined as follows. A *cut* of F for P is a position (i, j) such that $i, j \in F$ and the position is not covered by any fragment in F.

We consider a PQ-tree shown in Fig. 3.3 (c). It consists of only one P-node and leaves. The lengths of fragments are $2, 3, 3, 3, \ldots, 3, 3, 3, 2$. The numbers of 1s assigned to each probes are $1, 2, 1, 2, \ldots, 2, 1, 2, 1$. The set of fragments covers the set of all probes. If additional fragment set doesn't cover the set of all probes as Fig. 3.3 (d), the fixing operation can't be completed. However, if this is a subtree of the given PQ-tree, although the additional fragment set doesn't cover the set of all probes, there is a case that the fixing operation can be completed. In many cases, it causes to save the total length of additional fragments.

For example, the additional fragment set shown in 3.3 (g) doesn't cover all probes by a path in the subtree whose root is v_2. However, the subtree is also fixed by assigning fragments to A and I which are next to the subtree as in the figure. Here, we pay attention only to the subtree v_2 (the subtree rooted by v_2) of Figures 3.3 (e)–(g). If additional fragments on the subtree are given as Fig. 3.3 (g), order of BCDE, E, and FGH cannot be fixed yet. Thus fragments A and I, which are neighbors of the subtree, must be covered by fragments. In other words, a naive procedure such as to find the optimal solution in each subtree and to build them up from leaves to the root simply may not to obtain an optimal solution.

For example, if the cut is moved to F as Fig. 3.3 (f), the total length increases from

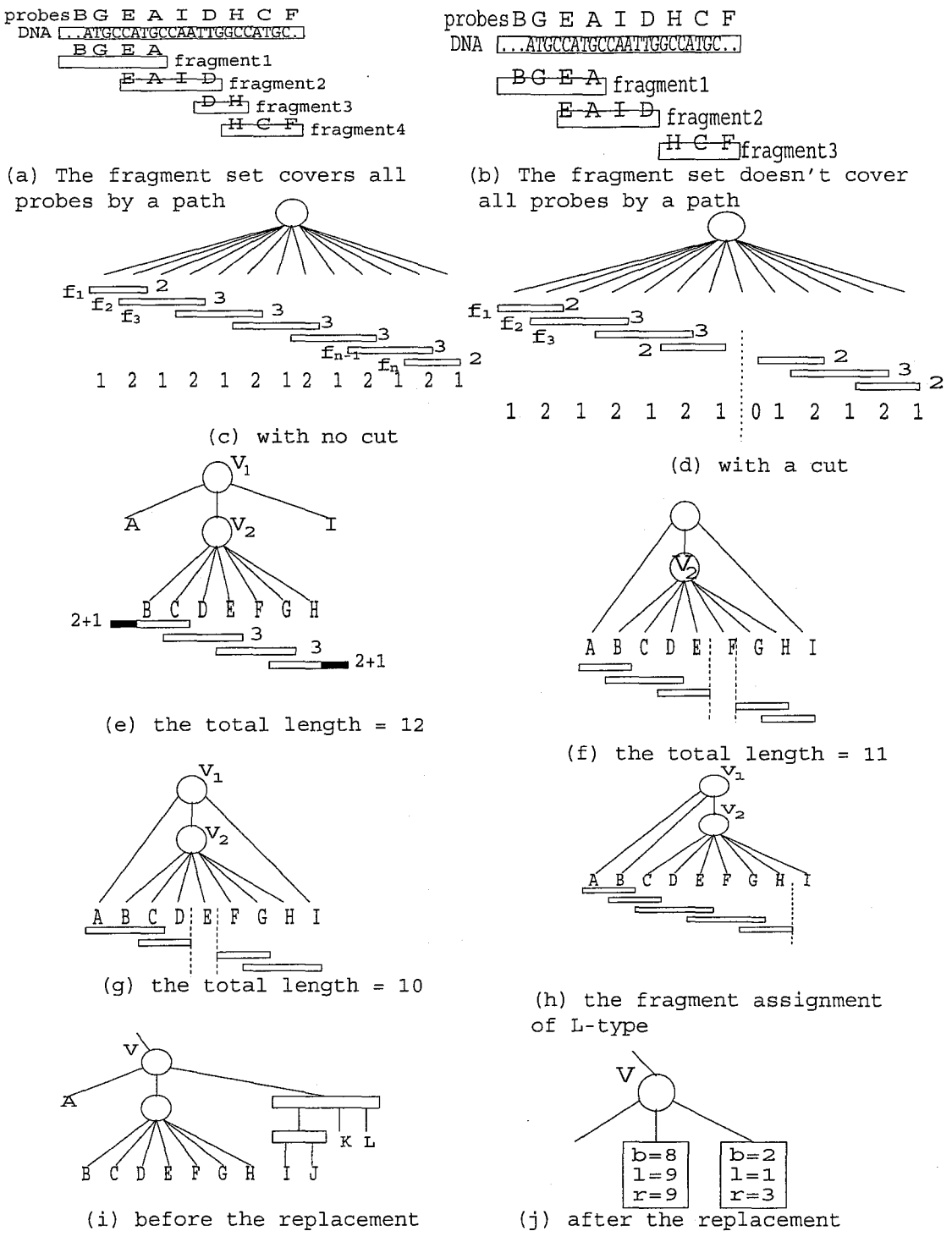


Figure 3.3: Minimizing the total length of additional fragments

10 to 11.

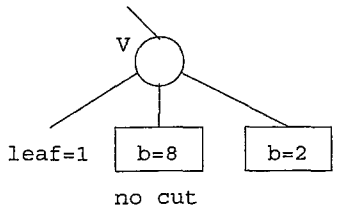
Hence, let us say that the pair of such subtree and such fragment assignment is B-type (B means "both sides".The precise definition will be done later). Moreover, if the fragment assignment on the subtree v_2 are given as Fig. 3.3 (h), we have to assign a fragment to B which is the left neighbor of the subtree. Hence, let us say that the pair of such subtree and such fragment assignment is L-type. R-type is defined symmetrically. More precisely, they are defined as follows: (Note that a pair of a subtree and a fragment assignment can be two or three types at a time.)

- **R-type** A pair of a subtree and a fragment assignment, such that if there is 1 at the right neighbor probe of the subtree, the subtree can be transformed into 1Q-tree and connected to the right side.
- **L-type** A pair of a subtree and a fragment assignment, such that if there is 1 at the left neighbor probe of the subtree, the subtree can be transformed into 1Q-tree and connected to the left side.
- **B-type** A pair of a subtree and a fragment assignment, such that if there are 1s at the both neighbor probes of the subtree, the subtree can be transformed into 1Q-tree and connected to the both sides.

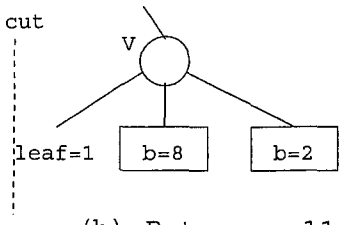
The minimum value of the total length of feasible fragment assignments for each of the three types can be calculated in polynomial time, since if the cut is fixed, then the minimum value can be obtained easily. By memorizing the minimum values of the total length of additional fragments for each of the three types for every subtree, we can also calculate the minimum values of them for the upper subtrees. Now, we establish an algorithm, which examines all candidates of the cut and finds the optimal fragment assignments in the three types for every sub-tree, in order to find the minimum fragment set of the whole PQ-tree.

The following example explains the algorithm more in detail. Fig. 3.3 (i) can be replaced with Fig. 3.3 (j) by calculating the optimal fragment assignments for the three types for every subtree except for v. Let b, l and r be the minimum values for the total lengths of the additional fragment sets of B-type, L-type and R-type, respectively.

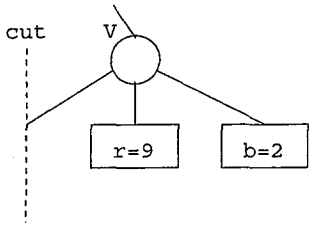
How to calculate b,l, and r



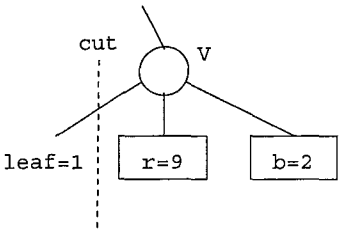
(a) B-type = 11



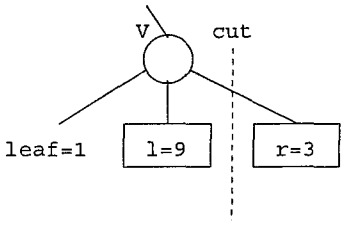
(b) R-type = 11



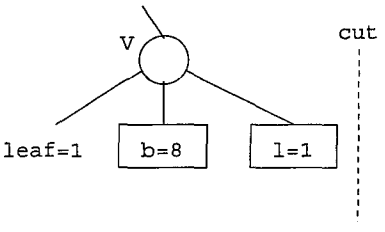
(c) R-type = 11



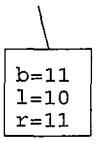
(d) B-type = 12



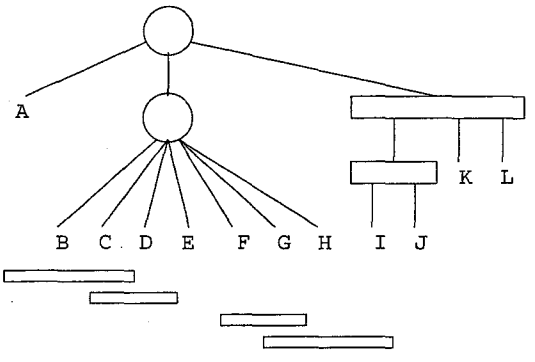
(e) B-type =13



(f) L-type = 10



(g) The full replacement



(h) The minimum fragment set which is necessary for the fixing operation

Figure 3.4: Calculating b,l and r

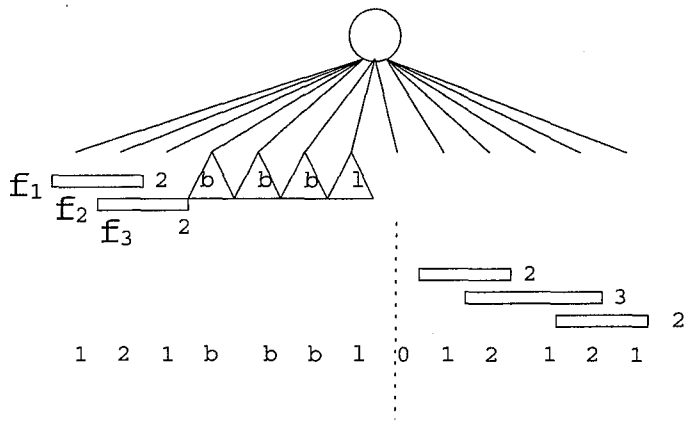


Figure 3.5: Assignment for a subtree which has child nodes and the root is a P-node

From figures 3.4 (a) to (f), the cut is moved from the left side to the right side, in order to examine every case. When the location of the cut is fixed, the optimal fragment assignment and the type, B, L, or R, are automatically calculated. Moreover, in the Fig. 3.4 (g), we look for the optimal fragment assignments for the three types, and replace with Fig. 3.3 (j). In this way, we gradually investigate the optimal fragment assignments in the three types for upper subtrees.

Our algorithm processes every internal node. For every two internal nodes which are in parent-child relation, the child node must be processed before the parent node, e.g., the post-order of DFS. The processes for subtrees in which the roots are P-nodes are simply explained by using the examples above. Details are as follows:

1. processes for each subtree whose root is a P-node

   (a) fix a cut

   A root P-node can have at most one cut. The cut is located either on a child of the P-node or on a space between children of the P-node. Let $p_1, p_2, \ldots, p_d$ be the children of the root P-node. The cut can be located at $p_1, p_2, \ldots, p_d$, $(-, p_1)$, $(p_1, p_2), \cdots, (p_{d-1}, p_d)$ and $(p_d, -)$.

   (b) algorithms for leaves that are children of the root P-node

   Assign fragments whose lengths are $2, 3, 2, 3, \ldots, 2$ as shown in Fig.s 3.3 (c) or (d). The number of 1s that are assigned to each fragment is $1, 2, 1, 2, \ldots, 1, 2$.

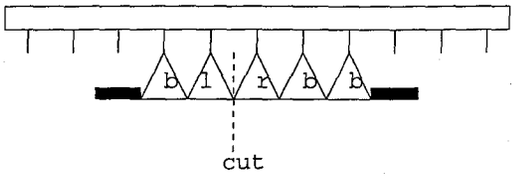


Figure 3.6: Assignment for consecutive children nodes of a Q-node

(c) algorithms for internal nodes that are children of the root P-node.

Since the cut is already fixed at this point, the type of each subtree whose root is a child of the P-node is automatically determined. For example, in Fig. 3.5, the subtree that is the left neighbor of the cut should be assigned L-type. The other subtree that are children of the P-node should be assigned B-type.

The processes for each subtree whose root is a Q-node are as follows: Let $p_1, p_2, \ldots, p_d$ be children of the Q-node.

1. the processes for p_1 and p_d.

   Since processes for p_d can be applied for p_1, we discuss only about p_d. When p_d is a leaf node, we examine both 0 and 1. When p_d is an internal node, we examine R-type, L-type and B-type.

2. The processes for $p_2, p_3, \ldots, p_{d-1}$

   Let p_i be the root of the subtree that we want to assign fragments.

   (a) the processes for the part in which internal nodes are consecutive

   We assign R-type, L-type or B-type according to the location of the cut. When there is a cut on the right neighbor of the part, we assign the L-type. When there is a cut on the left neighbor of the part, we assign the R-type. In the other cases, we assign the B-type. For example, in Fig. 3.6, the left neighbor node of the cut is assigned L-type, the right neighbor node of the cut is assigned R-type, and the other nodes are assigned B-type.

   (b) **The processes for the part in which there are internal nodes and leaves alternately**

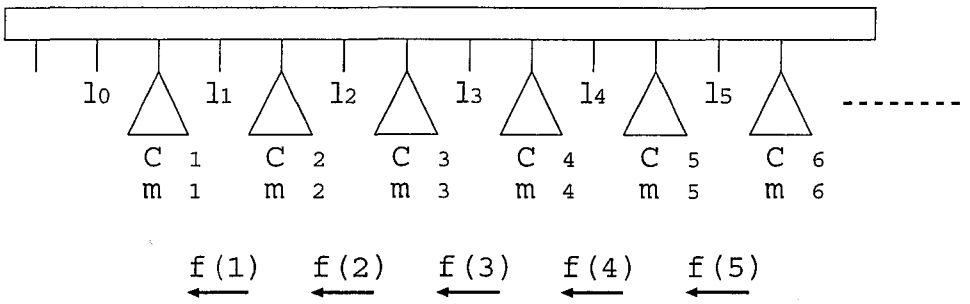


Figure 3.7: A Q-node which has a part in which there are internal child nodes and leaves alternately

We assign the minimum assignment of R-type, L-type and B-type, from the left side to the right side. If there are two types in which the total lengths are the minimum, we use the following criteria:

- **State 1 (the initial state)** The priority order is as follows: R-type, L-type and B-type. When we assign R-type or B-type, we go to state 2. When we assign L-type, we remain state 1.
- **State 2**
  The priority order is as follows: L-type, R-type and B-type. When we assign L-type, we go to state 1. When we assign R-type or B-type, we remain state 2.

(c) **The processes for the part in which there are two leaves between the internal nodes** We can treat them separately.

By using the algorithm, we obtain the following theorem.

Theorem 3.1 *A fragment set for* $FIX(T, \sigma, \Sigma)$ *can be found in* $O(n^2)$ *time, where* n *is the number of probes.*

Proof.

When the minimum fragment set is assigned to each subtree, we can determine whether each subtree and fragment assignment are L-type or not, whether each subtree and fragment assignment are R-type or not, and whether each subtree and fragment

assignment are B-type or not. If a subtree and corresponding fragment assignment do not belong to either L-type, R-type, or B-type, the fixing operation cannot be completed. Then, our dynamic programing algorithm can find the minimum fragment set correctly. The proof for the computational time is as follows. Let d_i be the degrees for each internal nodes x_i. Since, the computational time for each internal node is at most $O({d_i}^2)$, the whole computational time is at most $\sum_{x_i} O({d_i}^2) = O(n^2)$. □

3.2.2 $FIX(T, -, \Sigma)$

In $FIX(T, -, \Sigma)$, since there is no distinction between L-type and R-type, they are called *LR-type*. Let lr be the smaller one of l and r. Although, in $FIX(T, \sigma, \Sigma)$ a cut is scanned from left to right, in $FIX(T, -, \Sigma)$ a cut is fixed. However, the algorithm has to examine all candidates of nodes for both adjacent sides of the cut and the leftmost node and the rightmost node of the subtree. For the other nodes, the B-type assignment in which the total length of fragments is less than any other B-type assignment is used. Since the position of the cut and whether there is a cut or not are assumed in advance, the algorithm is not allowed to make a new cut by assigning fragments to nodes. However, if the B-type assignment is replaced by another assignment which is not B-type, a new cut is created.

Leaves and internal nodes should be ordered alternately as far as possible. Although the algorithm has to examine more cases, the order of the computation time doesn't become large.

Theorem 3.2 *A fragment set for $FIX(T, -, \Sigma)$ can be found in $O(n^5)$ time, where n is the number of probes.*

Proof.

Let d_i be the degrees for each internal nodes x_i. The number of combinations of candidates for both adjacent sides of the cut and the leftmost node and the rightmost node of the subtree is $O({d_i}^4)$. Since the algorithm has to scan the subtree in order to assign fragments, the computational time for each internal node is $O({d_i}^5)$, Then, the whole computational time is at most $\sum_{x_i} O({d_i}^5) = O(n^5)$. □

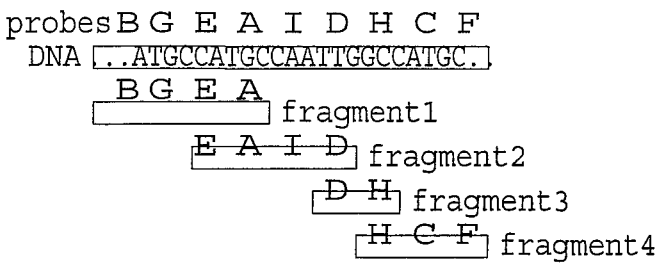


Figure 3.8: In case a fragment set covers all probes by a path

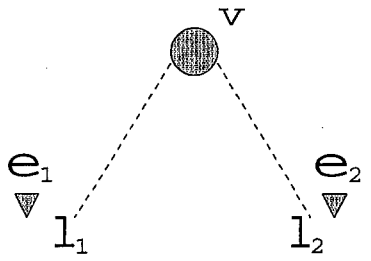


Figure 3.9: The nearest common ancestor

3.3 Minimizing the Number of Additional Sub-Fragments

In this section, additional fragments are limited to sub-fragments of existed fragments for fixing operations. The criterion is still to minimize the number of additional fragments. What is the difference between the general case and in this section? It is clear that more additional fragments are needed in this case than the general case. We discuss about how many fragments are more needed in this section than in the general case.

Now, we define to cover all probes by a path. Let (i, j) be neighbor probes. When a fragment set is given, if there is a fragment that includes both of i and j on every (i, j), let us say that the fragment set *covers all probes by a path.* For example, in Fig. 3.8, the fragment set covers all probes by a path.

Next, we define the nearest common ancestor node of two edges. Suppose that a fragment is terminated by two edges, e_1 is the left edge and e_2 is the right edge. Let l_1 be the right probe of e_1, and l_2 as the left probe of e_2 (See Fig. 3.9). When v is the

nearest common ancestor of l_1 and l_2, v is also called *the nearest common ancestor of* e_1 *and* e_2. The ancestor nodes of l_1 are called the ancestor of e_1. The ancestor nodes of l_2 are called the ancestor of e_2.

In Chapter 2, two edges are disappeared by an additional fragment that is constructed by the method of Lemma 2.3. The next lemma shows that this method can be used also in this section. However, existing fragments should include at least 3 edges.

Lemma 3.1 *If a fragment set can cover all probes by a path and there is a fragment that includes at least 3 edges, the number of edges can be decreased by 2 by an additional sub-fragment.*

Proof.

It can be proved by Lemma 2.3 and Theorem 2.1. □

If there is no existing fragment that includes 3 edges, the method of Lemma 2.3 cannot be used any longer. However, the next lemma shows that even if there is no fragment that includes 3 edges, the number of edges can be decreased at least by one.

Lemma 3.2 *If a fragment set can cover all probes by a path and there is a fragment which includes 2 edges, the number of edges can be decreased at least by one by an additional fragment.*

Proof.

1. when the distance of two edges is greater than or equal to 2

   The number of edges can be decreased by at least 1 by an additional fragment terminated by these two edges.

2. when the distance of two edges is 1

   Suppose that there are two Q-type edges that are neighbors each other. It is not necessary to use both edges in the fixing operation. Since there is a fragment

set that covers all probe by a path, there is no Outside-P-type edge. Then, both edges are Inside-P-type.

Since there are 2 edges, the length of the existing fragment is greater than or equal to 3. Then, a new sub-fragment that is terminated by an edge and that the length is greater than or equal to 2 can be constructed. Because of assumptions of templates, an Inside-P-type edge can be disappeared by an additional fragment that is terminated by the edge and that the length is greater than or equal to 2.

Thus, the lemma is proved. □

The previous lemma showed that when there is an existing fragment that includes 2 edges, the number of edges can be decreased at least by one by an additional sub-fragment. Next, we discuss about the case that all fragments include at most 1 edge. The next lemma shows that all edges can be decreased by adding sub-fragments.

Lemma 3.3 *Suppose that there is a fragment set that covers all probes by a path. If every existing fragment has at most one edge, the fixing operation can be done by only adding sub-fragments.*

Proof.

1. When the edge is Inside-P-type.

   The length of an existing fragment that includes an edge is greater than or equal to 2 from the assumption.

   (a) When the length of the existing fragment that includes an edge is 2:

      i. When the number of children of the parent vertex of the edge is 2:
      A P-node that has only 2 children can be treated as a Q-node.

      ii. When the number of children of the parent P-node of the edge is more than 2:
      Since there is a fragment set that covers all probes by a path, there is an existing fragment that is terminated by the edge. When the number of children of the parent node of an Inside-P-type edge is greater than 2, the edge should have been disappeared by the fragment terminated by the edge. Thus, this case does not occur.

(b) When the length of the existing fragment that includes the edge is greater than 2:

i. When the number of children of the parent node of the edge is 2:
The P-node can be treated as a Q-type node.

ii. When the number of children of the parent node is greater than 2:
A sub-fragment that the length is 2 and that is terminated by the edge can be constructed. The edge can be disappeared by this fragment.

2. When the edge is Q-type.

If there are sub-fragments that cannot disappear any edges but transform the PQ-tree, we continue to add such fragments. When templates Q2 or Q3 are applied, the number of nodes decreases. Q-type edges never become Inside-P-type or Outside-P-type. If the edge disappears or the 1Q-tree is gotten, our aim is achieved. If the edge does not disappear and the 1Q-tree is not gotten, the same situation remains. Since the number of nodes is finite, this situation does not continue infinitely. Then, the situation occurs in which it is impossible to add sub-fragments that can transform the PQ-tree.

If an additional fragment is terminated by the Q-type edge and another end is outside of the parent Q-node, the PQ-tree is transformed. Then, this case does not occur. At this point, every probe in this existing fragment is child of the same Q-node. Then, this Q-type edge can be moved into the overlapping part of this fragment and the neighbor fragment. If a fragment which includes two edges is created by this movement, the same process of the former case should be applied. By using the same discussion in the neighbor fragment, this edge will be disappeared or the 1Q-tree is gotten by the spreading of this Q-node.

3. When the edge is Outside-P-type.

Since there is a fragment set which covers all probes by a path, there are no Outside-P-type edges.

□

Thus, the previous lemma showed that the number of edges always can be decreased at least by one by an additional sub-fragment. Since the number of edges is finite, the fixing operation can be done by adding finite sub-fragments. The next theorem shows that the fixing operation can be done by only sub-fragments.

Theorem 3.3 *When there is a fragment set which covers all probes by a path, the fixing operation can be done by only adding sub-fragments of existing fragments.*

Proof.

From Lemmas 3.1 and 3.2, the number of edges can be decreased to less than or equal to 1. Then, this theorem can be proved by Lemma 3.3. □

Every Q-node that does not have child nodes has Q-type edges. When an additional fragment is added, there is a possibility that a Q-node spread, that is, the number of probes that are children of the Q-node is increased. An additional fragment should be terminated by spaces that one is under the Q-node and the other is outside of the Q-node in order to spread the Q-node as shown in Fig. 3.10. In the next lemma, we discuss about the number of probes that are children of a Q-node that is spread by an additional fragment.

Lemma 3.4 *Let v be the nearest common ancestor of the left end and the right end of the additional fragment. When a fragment that is terminated by a Q-type edge and the other end is outside of the parent Q-node (See Fig. 3.10), the probe set under the Q-node is as follows after the PQ-tree is transformed:*

1. *When v is a P-node.*

   Let v_l be the node that is the ancestor of the left end of the fragment and the child of v. Let v_r be the node that is the ancestor of the right end of the fragment and the child of v. Let l_1 be the leftmost probe included in the subtree whose root is v_l. Let l_r be the rightmost probe included in the subtree whose root is v_r. The probe set whose leftmost is l_l and rightmost is l_r will be under the Q-node.

2. *When v is a Q-node.*

   Every probe under the subtree whose root is v will be under the Q-node v.

Proof.

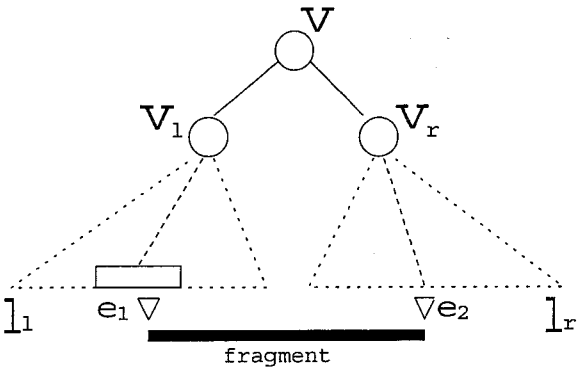


Figure 3.10: An additional fragment that is terminated by a Q-type edge and the other end is outside of the parent Q-node

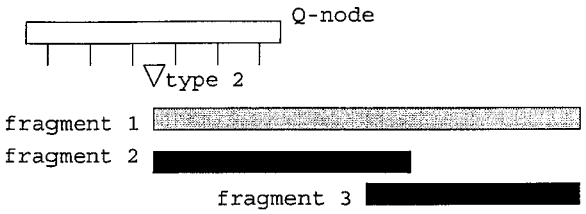


Figure 3.11: Dividing the additional fragment to a Q-node

Let v_q be the Q-node that is the parent of the left end of the additional fragment. Suppose that $v_1, v_2, \ldots, v_k$ be the nodes that are influenced by the additional fragment. The nearest common ancestor of $v_1, v_2, \ldots, v_{k-1}$, and v_k is v. When a fragment is added, templates are applied to $v_1, v_2, \ldots, v_{k-1}$, and v_k from bottom up. Let v_i be the node that a template is going to be applied.

1. when $v_i \neq v$

   When templates are applied except for v, P2, P4, P6 and Q3 can't be applied. Then, only P3, P5 and Q2 can be applied. Suppose that either P3, P5 or Q2 is applied to the PQ-tree. Let $p_1, p_2, \ldots, p_d$ be probes that are children of v_i. $p_1, p_2, \ldots, p_d$ will be the children of v_q after the template is applied.

2. when $v_i = v$

   At this point, templates for $v_1, v_2, \ldots, v_k$ except for v are finished to be applied. Every probe that was included in the subtree whose root was v_l is a descendant of the v_l. Moreover, v_l is a Q-node. The same discussion can be applied to v_r.

   Since an end of the additional fragment was Q-type edge, either v_l or v_r is partial. Thus, either P2 or P3 can't be applied. Since every 1 is under v, P5 can't be applied. Thus, only P4, P6, Q2 and Q3 can be applied. When either P4 or P6

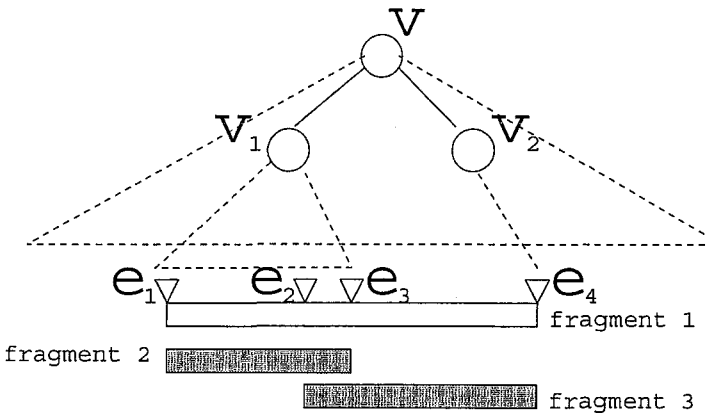


Figure 3.12: The nearest common ancestors of additional fragments

is applied, the condition is satisfied. Similarly, when either Q2 or Q3 is applied, the condition is satisfied.

□

Lemma 3.4 shows the number of probes that are descendant of the Q-node that is spread by an additional fragment. Next, we discuss about the case in which the fragment is divided into 2 as shown in Fig. 3.11. The next lemma shows that 2 divided additional fragments can make the same probe set that are descendant of the Q-node.

Lemma 3.5 *Suppose that an end of additional fragment is Q-type edge and the other end is outside of the parent Q-node. When the additional fragment is divided into 2 with a overlapping, the same probe set will be under the parent Q-node after the PQ-tree is transformed.*

Proof.

The fragment that is not divided is called fragment 1. Let e_1 and e_4 be ends that terminate fragment 1 as shown in Fig. 3.12. The fragments that can be made by dividing fragment 1 are called fragment 2 and fragment 3. Let e_1 and e_3 be ends that terminate fragment 2. Let e_2 and e_4 be ends that terminate fragment 3. Let v be the nearest common ancestor of e_1 and e_4.

1. when v is a Q-node

   When fragment 1 is added, every probe that was descendant of v will be the descendant of v from Lemma 3.4. Let v_1 be the nearest common ancestor of e_1 and e_3. Let v_2 be the nearest common ancestor of e_2 and e_4.

   (a) when $v = v_1$

   Let p_{23} be the probe set that are descendant of the Q-node after adding fragments 2 and 3. Let p_1 be the probe set that are descendant of the Q-node after adding fragment 1. When fragment 2 is added, every probe that was descendant of v will be the descendant of the Q-node. At this point, e_1, e_2, e_3, and e_4 are descendant of the Q-node. Then, when fragment 3 is added, the number of probes that are descendant of the Q-node is not changed. Thus, $p_{23} = p_1$.

   (b) when $v \neq v_1$

   If e_4 was a descendant of v_1, v_1 is the nearest common ancestor of e_1 and e_4. Although v_1 is a child of v, v is the nearest common ancestor of e_1 and e_4. It is a contradiction. Thus, e_4 is not in the subtree whose root is v_1. An end that terminates fragment 3 is under the Q-node. The other end is outside of the Q-node. Since v is a Q-node, every probe that was a descendant of v will be a descendant of the Q-node after adding fragments 2 and 3. Thus, $p_{23} = p_1$.

2. when v is a P-node

   Suppose that fragment 1 is added. From Lemma 3.4, the leftmost child of the Q-node is the leftmost probe of the subtree whose root is v_1. Similarly, the rightmost child of the Q-node is the rightmost probe of the subtree whose root is v_2. Since v_1 is the nearest common ancestor of e_1 and e_3, v_1 is the final node that a template is applied after adding fragment 2. Then, additional fragment 2 never change v. Then, v is still a P-node. At this point, v_l is the ancestor of e_2 and the child of v. v_2 is the ancestor of e_4 and the child of v. The probe set which comes under the Q-node by additional fragment 3 is the same as by additional fragment 1.

□

Suppose that fragment 1 is efficient for a fixing operation, that is, the fragment can disappear 1 or 2 edges. Suppose that fragments 2 and 3 can be made by dividing fragment 1 and they have an overlapping part. The next lemma shows that it is also efficient for a fixing operation to add fragments 2 and 3 instead of fragment 1.

Lemma 3.6 *When an edge is disappeared by an additional fragment, 2 divided fragments whose lengths are at least 2 can disappear the same edge.*

Proof.

1. when the edge is Inside-P-type
   (a) when the degree of parent vertex is 2
   A P-node whose degree is 2 can be treated as a Q-node whose degree is 2. Then, we treat it as a Q-type edge.
   (b) when the degree of parent vertex is equal to or more than 3
   Let d_{v_p} be the degree of the parent vertex. If d_{v_p} is equal to or more than 3 and the additional fragment whose length is at least 2 is terminated by the Inside-P-type edge, the Inside-P-type edge will disappear.
2. when the edge is Q-type
   The parent P-node of the edge includes same probe set before dividing and after dividing from Lemma 3.5. There are two methods in order to disappear the Q-type edges.
   (a) the case in which the Q-node has a new internal child node (as templates P4, P5 and P6)
   Additional fragment 1 cannot disappear e_2 and e_3. However, fragment 3 can disappear e_2 and e_3. After the PQ-tree is converted, e_1, e_2, and e_3 are children of the parent Q-node of e_1. If e_1 is not disappeared and either e_2 or e_3 is disappeared, there is a possibility that either e_2 or e_3 is a Q-type edge.
   (b) when the Q-node includes another Q-type edge (as template Q3)
   After fragment 1 is added, there are no edges between these two edges. Then, neither e_2 nor e_3 is disappeared. Thus, both of these Q-type edges

remain. If e_1 is disappeared and e_4 is not disappeared, e_1 can be moved to e_4.

3. when the edge is Outside-P-type

   It is clear.

□

Theorem 3.4 *Suppose that there is a fragment set which consists of n fragments. If it can cover all probes by a path and the number of edges is e, the fixing operation can be done by adding* $\lceil \frac{e}{2} \rceil + n$ *sub-fragments of existing fragments.*

Proof.

From Lemmas 3.1, 3.2 and 3.6, and Theorems 2.1 and 3.3. □

In this section, only sub-fragments of existing fragments can be used. It is clear that this case needs more fragments to fix a PQ-tree than the general case. The problem was how many fragments are more necessary to fix a given PQ-tree. The algorithm has to investigate how many existing fragments are needed to cover all probes by a path. If there is a fragment set which can cover all probes by a path, the fixing operation can be done by $\lceil \frac{e}{2} \rceil + n$ additional fragments. If any fragments can be used, the fixing operation can be done by $\lceil \frac{e}{2} \rceil$ additional fragments. Then, n fragments are more needed in the restriction of this section. Moreover, we can make the minimum fragment set in polynomial time from the same reason of the general case.

3.4 Minimizing the Number of Additional Fragments of Length Two

In this section, lengths of additional fragments are limited to 2 in the fixing operation. Since an additional fragments whose length is 1 cannot transform a PQ-tree, a fragment whose length is 2 is the shortest fragment for fixing operations.

First, we discuss the case in which the simplest PQ-tree is given as shown in Fig. 3.13. The next lemma shows that what kind of additional fragments whose lengths are 2 are necessary to fix the simplest PQ-tree.

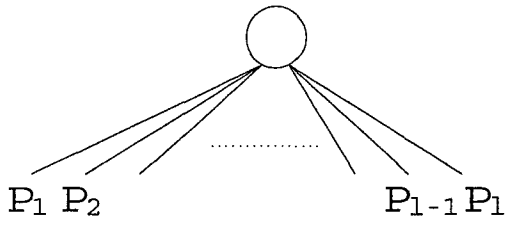


Figure 3.13: The simplest PQ-tree

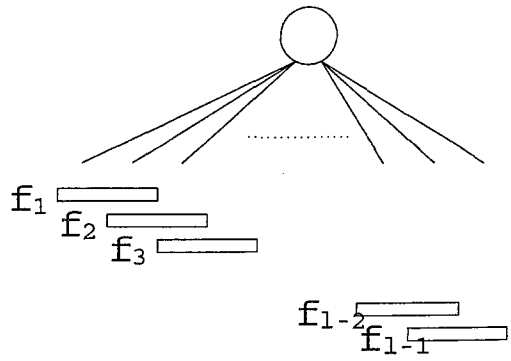


Figure 3.14: Additional fragments for the simplest PQ-tree

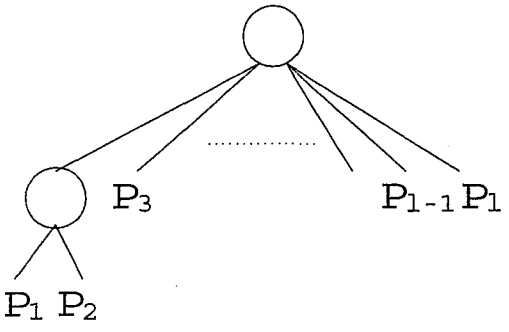


Figure 3.15: A PQ-tree that has 2 P-nodes

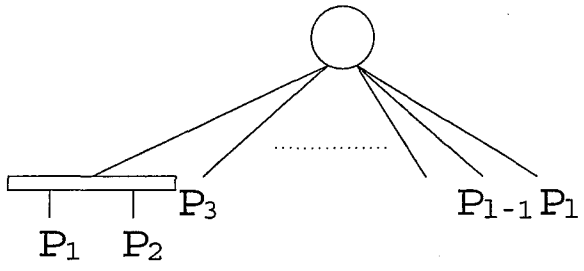


Figure 3.16: Only one method to add $l-1$ fragments

Lemma 3.7 *When a PQ-tree which has l leaves and one root P-node is given ($l \geq 3$),*

1. *the fixing operation can be done by adding $(l-1)$ fragments whose lengths are 2.*
2. *the fixing operation cannot be done by adding less than $(l-1)$ fragments whose lengths are 2.*

Proof.

1. The lengths of additional fragments are limited to 2. Thus, if the left end of a fragment is decided, the right end is automatically decided. Since the given

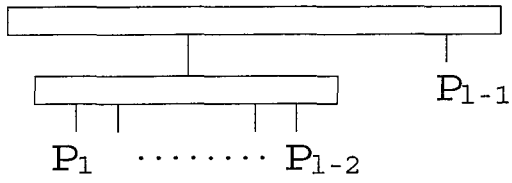


Figure 3.17: After f_{l-2} is applied

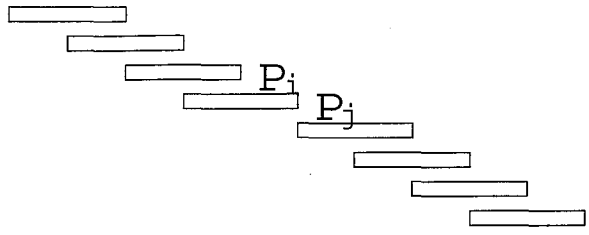


Figure 3.18: When there is a cut between p_i and p_j

PQ-tree has l leaves and one P-node, there are only $(l-1)$ ways to add fragments as shown in Fig. 3.14.

Since it is meaningless to add same fragments, the method for adding $(l-1)$ fragments whose lengths are 2 is uniquely determined. Let $f_1, f_2, \ldots, f_{l-1}$ be the additional fragments for the simplest PQ-tree as shown in Fig. 3.14. When the first fragment f_1 is added to the PQ-tree, the template P2 is applied and the converted PQ-tree is as shown in Fig. 3.15.

A P-node that has 2 children can be treated as a Q-node that has 2 children. Thus, the PQ-tree as shown in Fig. 3.15 can be treated as the PQ-tree as shown in Fig. 3.16.

While fragments $f_2, \ldots, f_{l-2}$ are added, the template P4 is applied again and again. The number of probes that are children of the Q-node increases one by one. After f_{l-2} is added, the gotten PQ-tree is as shown in Fig. 3.17. After f_{l-1} is added, the template Q2 is applied and the 1Q-tree is gotten.

2. Since the minimum fragment assignment for the simplest PQ-tree is as shown in Fig. 3.14, $(l-1)$ fragments whose lengths are 2 are necessary to cover all probes by a path. If there is not an additional fragment that does not include both of p_i and $, p_j$, the sequence of the probe sets $p_1 \sim p_i$ and $p_j \sim p_l$ cannot be decided. There is a cut between p_i and p_j as shown in Fig. 3.18. Thus, $(l-1)$ fragments whose lengths are 2 are necessary to fix the PQ-tree. □

The previous lemma shows what kind of additional fragments whose lengths are 2 are necessary to fix the simplest PQ-tree. In the next lemma, we discuss about a PQ-tree that is a little more complicated than the simplest PQ-tree.

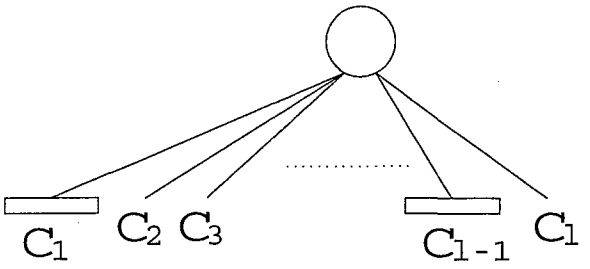


Figure 3.19: The second simplest PQ-tree

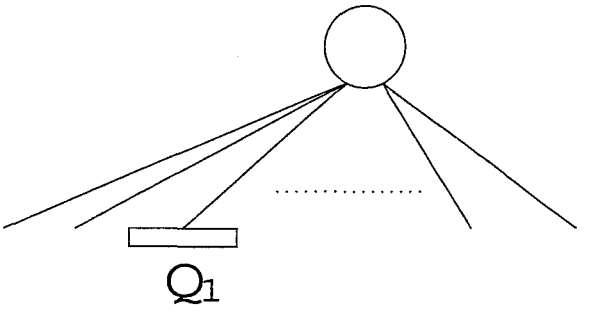


Figure 3.20: With a Q-node child

Lemma 3.8 *Suppose that a given PQ-tree whose internal nodes consist of 1 root P-node and Q-nodes that are children of the P-node. Let l be the number of children of the root P-node. When the length of additional fragments are 2,*

1. *the fixing operation can be done by* $(l-1)$ *additional fragments whose lengths are limited to 2.*
2. $(l-1)$ *additional fragments whose lengths are 2 are necessary to fix the PQ-tree.*

Proof.

1. when every child is a leaf

   The case is same as Lemma 3.7.

2. when there is a Q-node that is a child of the root node

   Let Q_1 be the only one children of the root node as shown in Fig. 3.20. An additional fragment in which an end is inside of Q_1 and the other end is outside of Q_1 should be added as shown in Fig. 3.21.

   (a) when the left neighbor probe of Q_1 is a leaf

   Let l_1 be the left probe of Q_1. Let l_2 be the leftmost probe of Q_1 as shown in Fig. 3.21. The fragment that includes Q_1 and Q_2 should be added. Then, the template P4 is applied and the converted PQ-tree is as shown in Fig. 3.22.

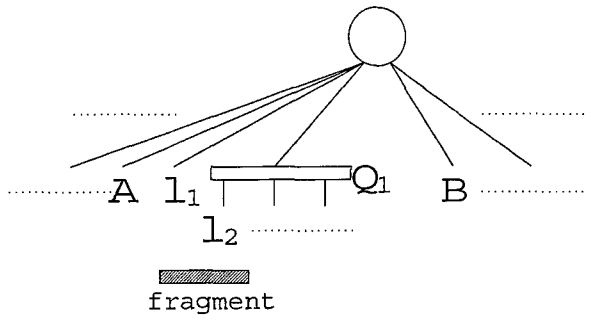


Figure 3.21: Adding a fragment from Q_1 to the left side

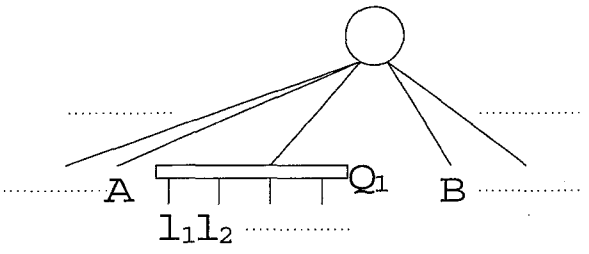


Figure 3.22: After adding the fragment

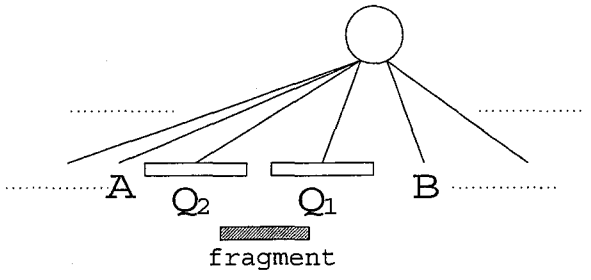


Figure 3.23: When the left side of Q_1 is a Q-node

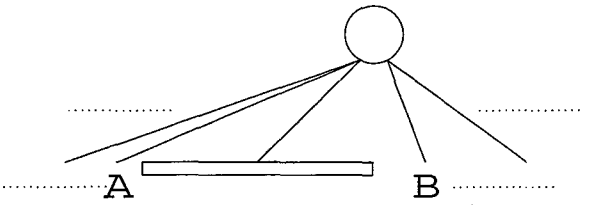


Figure 3.24: After adding the fragment

(b) when the left neighbor of Q_1 is a Q-node

Let Q_2 be the Q-node that is the left neighbor of Q_1. The fragment which includes the rightmost probe of Q_2 and the leftmost probe of Q_1 should be added. Then, the template P6 is applied and the converted PQ-tree is as shown in Fig. 3.24.

(c) when Q_1 does not have a left neighbor

Nothing should be done.

Similar operations can be applied to the right neighbors of Q_1. The 1Q-tree is gotten by adding $(l-1)$ fragments whose lengths are limited to 2 by this method. □

Lemmas 3.7 and 3.8 show what kind of additional fragments whose lengths are 2 are necessary to fix very simple PQ-trees. The two simple PQ-trees that we discuss in

previous lemmas do not include subtrees. When there is a subtree in a given PQ-tree, more complicated algorithm is necessary to make additional fragment sets. Since the lengths of additional fragments are limited to 2 in this section, adding $(l-1)$ fragments can fix the PQ-tree which has a root P-node and l leaves. Then, the total length of additional fragments which are necessary to fix the PQ-tree is $2(l-1)$. However, if any length of additional fragments can be used, the total length of additional fragments which are necessary to fix the PQ-tree is $\frac{3}{2}(l-1)$. In the former case, the lengths of additional fragments are all two. However, in the latter case, the lengths of additional fragments are two and three. A half of them are two, and the other half of them are three.

3.5 Concluding Remarks

First, for the problem of fixing the probe order of a given PQ-tree, we presented polynomial time algorithms that minimize the total length of additional fragments. We solved not only the problem of fixing probes in a given order, but also the problem to how to determine the best order of the probes. The idea of "edges" cannot be used any longer in these problems.

Second, we discuss problems in which additional fragments are limited to sub-fragments of existing fragments. Suppose that there is a fragment set which consists of n fragments. If it can cover all probes by a path and the number of edges is e, the fixing operation can be done by adding $\lceil \frac{e}{2} \rceil + n$ sub-fragments of existing fragments. We also mention problems in which lengths of additional fragments are limited to 2.

Chapter 4

Inferring Pedigrees from Genetic Distances

4.1 Introduction

A phylogenetic tree is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor. Various methods to infer phylogenetic trees from genetic information have been researched [3, 11, 24, 27, 31, 66, 70]. Since a phylogenetic tree doesn't include cycles, it cannot represent recombinations, that is, there are no vertices whose indegrees are more than one. Then, data structures which are slightly different from phylogenetic trees have been also researched recently [58].

Vertices of phylogenetic trees represent species. However, there are a huge number of individuals in each vertex and they have blood relationships each other. If all individuals are represented by vertices, indegrees of vertices should be two. However, since the number of individuals is finite, there must be a node whose indegree is one or zero. Moreover, there is a possibility that the number of vertices which don't have in-edges is more than one. Then, in this chapter, we discuss problems to infer directed graphs whose indegrees are at most two (We call them *pedigree graphs*). When genetic distances between any two individuals are given, we discuss problems (i) to enumerate all pedigree graphs which satisfy a given genetic distances. (ii) to validate whether

there is a contradiction or not between a given distances and a given pedigree graph. Genetic distances can be calculated from the number of differences of SNPs (Single Nucleotide Polymorphisms). If any distances between two adjacent vertices are given, the solution can be constructed very easily. Then, from a theoretical interest, we assume that a genetic distance is gotten as the shortest path of blood relationships.

Phylogenetic networks [75] are very similar to pedigree graphs. Then, it can be said that our problem is a special case of problems to infer phylogenetic networks. A phylogenetic network is a directed graph in which degrees of vertices are at most two and there is only one vertex whose indegree is zero. Problems to infer or compare phylogenetic networks have been researched recently: Inferring phylogenetic networks when 0-1 sequences are given for every species [15, 32, 75], constructing phylogenetic networks from phylogenetic trees [39, 42], finding similar sub-graphs from phylogenetic networks [18, 43], and so on. In these problems, they have a firm precondition that there is only one vertex whose indegree is zero in a phylogenetic network. However, in the problems to infer pedigree graphs, if an individual has no information about its ancestors, the indegree of the corresponding vertex should be zero. Then, in pedigree graphs, the number of vertices whose indegrees are zero may be more than one.

There are projects to develop drugs by determining genes which cause diseases from pedigrees and medical records [59]. However, there is a possibility that there are errors in given pedigrees. In this chapter, we also discuss problems to detect errors in pedigrees.

Distance Realization Problem is a problem to infer graphs or networks from given distance matrices [35]: problems to minimize the total length of edges [35], problems to minimize the number of adding vertices [6], problems in which solutions must be trees [28, 66], and so on. Most of Distance Realization Problems seem to allow adding new vertices. If adding new vertices is not allowed, one of the solutions is a complete graph which can be trivially made from the given distance matrix. However, if a triangle inequality is not satisfied, there are no solutions. If the length of the edge v_iv_j is equal to the sum of the lengths of the edges v_jv_k and v_kv_i in a solution, the graph gotten by removing the edge v_iv_j is also a solution. Our problem is regarded as a kind of Graph Realization Problems. Adding new vertices are not allowed in our problems. Solutions are directed graphs in which indegrees are at most two. The number of vertices whose indegrees are zero (or one) is not limited.

Many algorithms to generate a particular class of graphs, without repetition, are known[12, 54, 64]. If every solutions are really enumerated, the computation time should be at least the number of solutions. However, there is a case that computation time is drastically improved by using some techniques for compressing the representation of the solutions [65]. In this chapter, we discuss problems in which there is a possibility that the numbers of solutions are exponential of the inputs. However, since the outputs of the problems are defined as a digraph which represents a set of pedigree graphs which satisfy the given distance matrix, the computation times are polynomial.

Related Works

There are many papers that use pedigrees as inputs of problems. For example, the problem of haplotype inference under the Mendelian law of inheritance on pedigree genotype data is studied in [16] and [79]. The problem of finding a mini-mum-recombinant haplotype configuration (MRHC) is in general NP-hard [53]. [52] and [51] discuss about the case with missing alleles. Even if the input pedigree is loopless, MRHC is NP-hard [22]. [72] provides a tool for finding errors of given genotypes or phenotypes in a specified pedigree structure.

[1] deals with consistency checking problems as follows: given a pedigree and information on the genotypes of some of the individuals in it, the aim is to determine whether these data are consistent with the classic Mendelian laws of inheritance. There are also problems to count the number of complete consistent assignments. [69] provides polynomial time algorithms for loopless pedigrees. Pedigrees are treated as outputs in drawing and visualizing problems [76]. [71] provides a tool for drawing pedigrees with complex haplotypes.

4.2 Pedigree Graphs

Pedigrees are often represented as shown in Fig. 4.1. In this chapter, they are represented by directed graphs as shown in Fig. 4.2. A *pedigree graph* is a directed graph whose indegrees are at most two. The number of vertices whose indegrees are one or two is not limited.

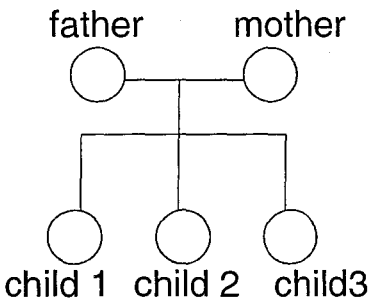


Figure 4.1: A pedigree

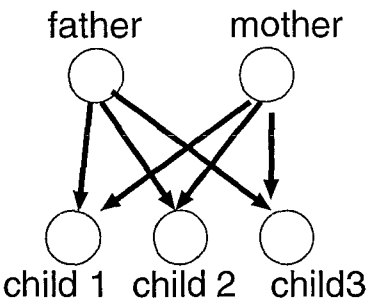


Figure 4.2: A pedigree written by a directed graph

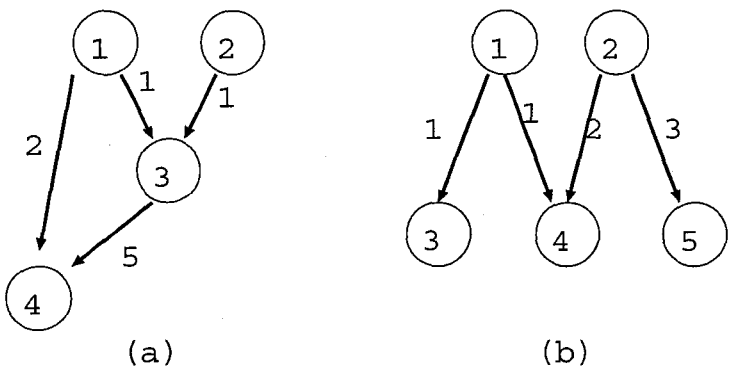


Figure 4.3: Direct distances and indirect distances

distance: The length of an edge of a pedigree graph represents the genetic distance between two vertices which are incident to the edge. For example, in Fig. 4.3 (a), the length of the edge between v_1 and v_4 is two (v_1 is denoted by 1 and v_4 is denoted by 4 for simplicity). Then, v_1 is one of the parents of v_4 and their genetic distance is two. Since we assume that a genetic distance is the shortest path of blood relationships, we define *distances* of pedigree graphs by using direct distances and indirect distances as follows: The *direct distance* between v_i and v_j is the length of the shortest directed path between v_i and v_j. The *indirect distance* between v_i and v_j is the minimum sum of two directed paths which are initialized by a common ancestor of v_i and v_j. For example, in Fig. 4.3 (a), the length of the shortest path between v_3 and v_4 is 5. Then, the direct distance between v_3 and v_4 is 5. On the other hand, v_3 and v_4 have a common ancestor

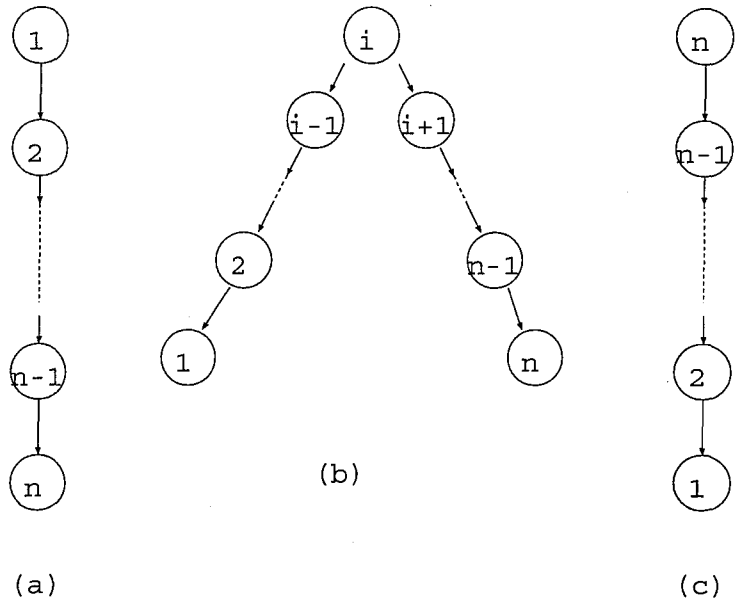


Figure 4.4: If a birth order is not given, they cannot be distinguised.

v_1. The length of the path between v_1 and v_3 is 1. The length of the path between v_1 and v_4 is 2. Then, the indirect distance between v_3 and v_4 is 1+2=3. The distance between v_i and v_j is defined as the smaller value of the direct distance of v_iv_j and the indirect distance of v_iv_j. Then, in Fig. 4.3 (a), since the indirect distance is shorter than the direct distance, the distance between v_i and v_j is 3.

In Fig. 4.3 (b), there is an undirected path between v_3 and v_5. However, v_3 and v_5 are not in blood relationships (i.e., they have no common ancestor).

distance matrix: When neither direct distances nor indirect distances exist, we say that the distance is ∞. From the definition, the distances between two vertices are symmetric. The matrix in which the element (i, j) corresponds the distance between v_iv_j $(i < j)$ is called a *distance matrix*. A distance matrix is denoted as shown in Fig. 4.6 (a) for simplicity.

birth order: When (i, j) $(i < j)$ of the given distance matrix is $j - i$, the pedigree graph as shown in Fig. 4.4 (a) saisfies the condition. However, pedigree graphs as shown in Fig. 4.4 (b) and (c) also satisfy the given distance matrix. In Fig. 4.4 (a), v_1 is the

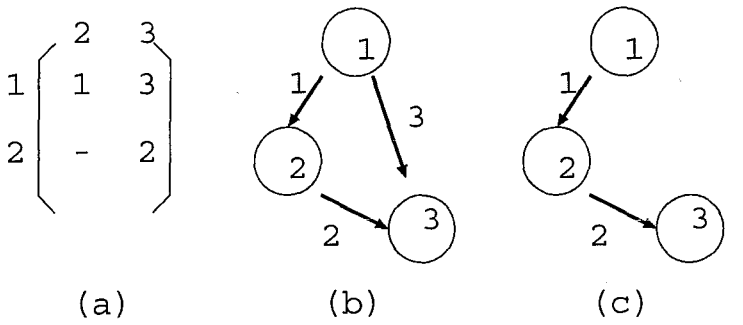


Figure 4.5: A redundant edge

oldest individual and v_n is the youngest individual. On the other hand, in Fig. 4.4 (c), v_n is the oldest individual and v_1 is the youngest individual. Thus, there is a case that even the oldest individual and the youngest individual cannot be distinguished. Then, we asuume that a birth order is given in our problems. When a *birth order* is given, if there is an edge between v_i and v_j $(i < j)$, v_i must be the parent of v_j. In the case of Fig. 4.4, if a birth order is given, Fig. 4.4 (a) is the only one solution. We use a birth order only for determining edge directions, that is, which is the parent and which is the child. Since a rigorous birth date or birth time is not needed, there are a lot of methods of getting a birth order.

redundant edges: The pedigree graph as shown in Fig. 4.5 (c) satifies the distance matrix as shown in Fig. 4.5 (a). However, the pedigree graph as shown in Fig. 4.5 (b) which can be made by adding an edge between v_1 and v_3 also satisfies the distance matirix. Moreover, if the length of the edge between v_1 and v_3 is 3 or more than 3, the pedigree graph satisfies the distance matrix. In such a case, it is impossible to determine the length of the edge between v_1 and v_3. When pedigree graphs $G_1 = (V, E)$ and $G_2 = (V, E - e_1)$ $(e_1 \in E)$ are given, if the corresponding matrices of G_1 and G_2 are same, e_1 is called a *redundant edge* and not included by solutions.

4.3 Pedigree Graph Enumerating Problem

At first we discuss the following trivial problem:

- Input: A distance matrix and a birth order.
- Output: A set of pedigree graphs, in which lengths of edges are one, which satisfy the distance matrix and the birth order.

Since lengths of edges are one, two vertices are adjacent if and only if the corresponding element of the distance matrix is one. Moreover, since the birth order is given, directions of edges are ditermined uniquely. Then, the solution is gotten in $O(n^2)$ time. The number of solutions is only one. Thus the case in which lengths of edges are one can be solved easily. Then, we discuss the case in which the lengths of edges are positive real values.

Pedigree Graph Enumerating Problem

- Input: A distance matrix and a birth order.
- Output: A set of pedigree graphs, in which the lengths of edges are positive real values, which satisfy the distance matrix and the birth order.

A directed graph in which there is just one edge between any two vertices is called a *tournament* [67]. When a distance matrix and a birth order are given, a tournament, in which weights are corresponding to the distance matrix and directions of edges are corresponding to the birth order can be uniquely obtained, and which is called a *basic tournament* and denoted by G_{in}. Moreover, a pedigree graph which doesn't contradict G_{in} and doesn't have redundant edges is called a *feasible pedigree graph* denoted by G_{out}. $\mathcal{G}_{out}$ is the set of G_{out}.

For example, G_{in} which corresponds to the distance matrix shown in Fig. 4.6 (a) is the tournament shown in Fig. 4.6 (b). Since both pedigree graphs shown in Fig. 4.6 (c) and (d) are consistent with G_{in}, they are elements of $\mathcal{G}_{out}$.

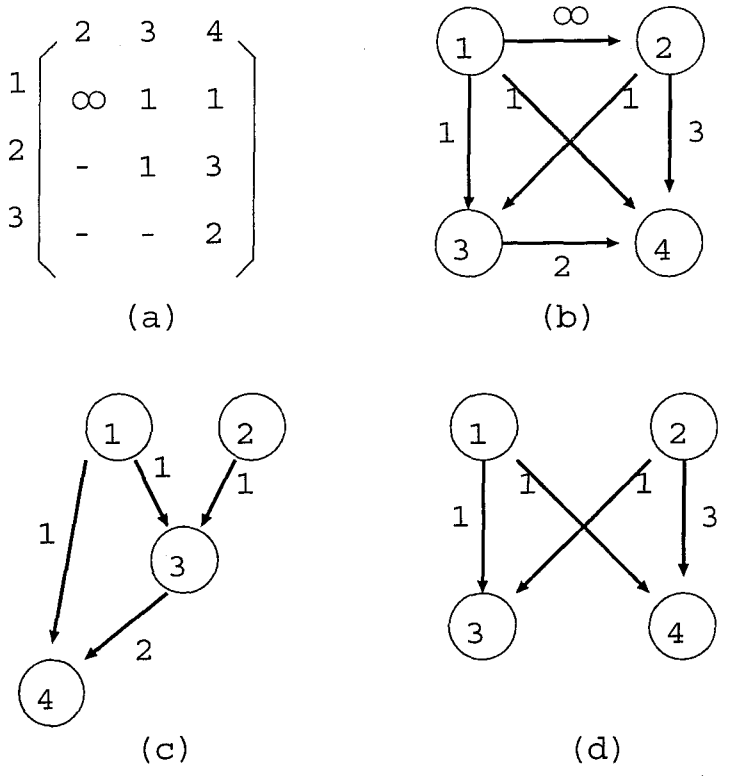


Figure 4.6: An example in which the solution can't be determined uniquely

We will show the following Theorem 2.1. For proving this theorem, we show some lemmas.

Let us denote the length of a directed edge (v_i, v_j) by "length(v_i, v_j)".

Lemma 4.1 *Let (v_j, v_i) be the shortest edge whose terminal vertex is v_i of G_{in}. Then a directed edge (v_j, v_i) also exists in G_{out}.*

Proof.

There is no edge which is shorter than (v_j, v_i) and is terminated by v_i in G_{out}. Assume that there is not (v_j, v_i) in G_{out}. Since the birth order is given, there is not (v_i, v_j). Since lengths of edges are positive, the distance between v_j and v_i is longer

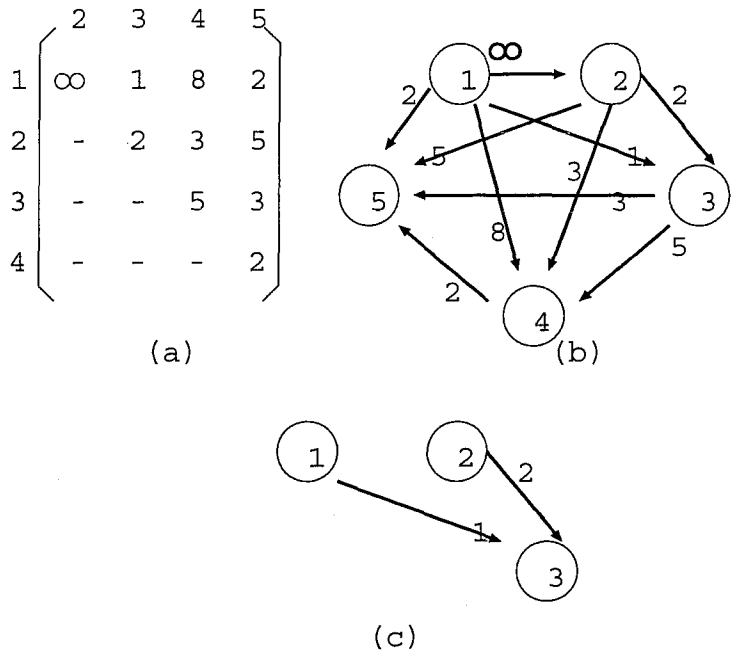


Figure 4.7: A distance matrix and the tournament

than length(v_j, v_i). It contradicts the assumption. Since the direction of the edge v_jv_i is determined uniquely, there is a directed edge v_jv_i in G_{out}. □

Lemma 4.2 *Let d be the length of the shortest edge terminated by v_i in G_{in}. Then, the number of edges which is terminated by v_i and whose length is d is at most two in G_{in}.*

Proof.

Asuume that there are three such edges. By Lemma 4.1, such edges also exist in G_{out}. Since indegrees of pedigree graphs are at most two, it is a contradiction. □

Lemma 4.3 *The indegree of v_j of G_{out} is zero if and only if all distances between v_iv_j $(i < j)$ are ∞ in G_{in}.*

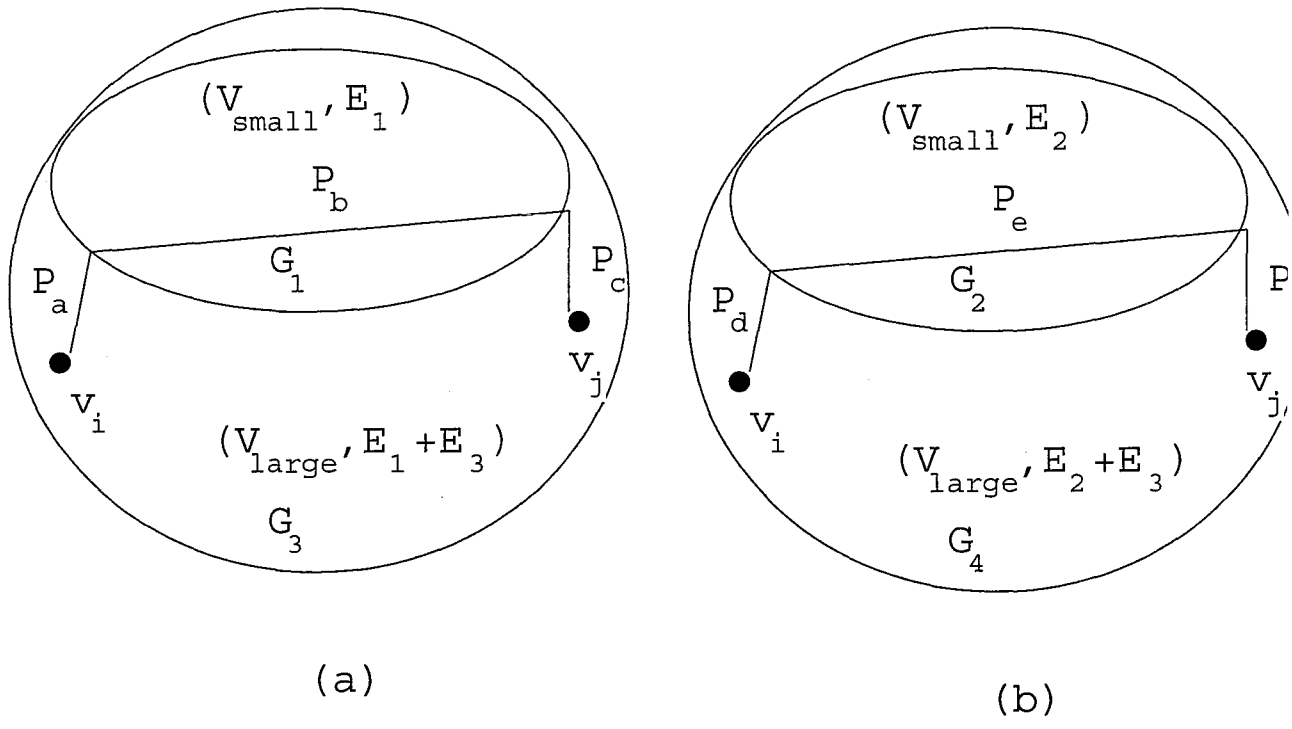


Figure 4.8: The relation among G_1, G_2, G_3 and G_4

Proof. Assume that the distance between v_iv_j is finite $(i < j)$. v_iv_j is either by the direct distance or the indirect distance. However, by the birth order, the path between v_i and v_j must include the edge terminated by v_j. Then, the indegree of v_j is not zero.

Moreover, if the distance between any v_iv_j $(i < j)$ is ∞, the indegree of v_j is zero$\square$

Theorem 4.1 claims that the Pedigree Graph Enumerating Problem can be solved in $O(n^3)$ time. The outline of the algorithm is as follows: At first, G_{in} is made from the given distance matrix. Next, G_{out} is gotten by removing unnecessary edges. We have to choose two edges for each vertices of G_{in} from $O(n)$ edges and use them as the edges of G_{out}. However, if all cases are explored, the exponential time will be needed.

For example, in a basic tournament shown in Fig. 4.7 (b), both edge pairs , $\{(v_1, v_4), (v_2, v_4)\}$ and $\{(v_2, v_4), (v_3, v_4)\}$, terminated by v_4 satisfy the distance matrix. A naive algorithm separating these two cases before the choosing operations at other vertices

may increase the computation time exponentially. However, the following lemma shows that the choosing operations can be done separately. Let $V_{small} = \{v_1, v_2, \cdots, v_n\}$ and $V_{large} = \{v_1, v_2, \cdots, v_{n+k}\}$. Moreover, let $G_1 = (V_{small}, E_1)$, $G_2 = (V_{small}, E_2)$ be pedigree graphs. Let length(P) be the length of a undirected path P. Let $v_{i'}, v_{j'} \in V_{small}$ and $v_{i''}, v_{j''} \in V_{large}$.

Lemma 4.4 *When distances between any $v_{i'}v_{j'}$ of G_1 and $v_{i'}v_{j'}$ of G_2 are same, for any E_3 which makes $G_3 = (V_{large}, E_1 + E_3)$ and $G_4 = (V_{large}, E_2 + E_3)$ pedigree graphs, distances between any $v_{i''}v_{j''}$ of G_3 and $v_{i''}v_{j''}$ G_4 are same.*

Proof.

The proof shown here is about when $v_i, v_j \in V_{large} - V_{small}$. This case can be applied to the other cases by putting ϕ to P_a, P_c, P_d and P_f.

Let $P_3 = P_a + P_b + P_c$ be the shortest path between v_iv_j in G_3. Let $P_4 = P_d + P_e + P_f$ be the shortest path between v_iv_j in G_4 (Fig. 4.8). Let $P_b \subseteq E_1$, $P_e \subseteq E_2$, and $P_a, P_c, P_d, P_f \subseteq E_3$. (There is a possibility that P_b and P_e are ϕ). Assume that length(P_3)$\neq$length(P_4). (The generality is not lost by length(P_3)>length(P_4)). By the assumptions of G_1 and G_2, there exists P_g, whose initial node and terminal node are same as P_e, which satisfies length(P_g)=length(P_e) and length(P_g)$\subseteq$ E_1. Then, $P_5 = P_d + P_g + P_f$ satisfies $P_5 \subseteq E_1 + E_3$ and length(P_5)<length(P_3). If P_5 is a direct distance of v_iv_j or an indirect distance of v_iv_j, it contradicts that length(P_3) is the shortest path of v_iv_j, and length(P_3) = length(P_4) can be proved.

The following proof shows that P_5 is a direct distance of v_iv_j or an indirect distance of v_iv_j.

- When P_3 is composed of a directed path:

  The generality is not lost by assuming that each of P_b, P_c is composed of a directed path and $P_a = \phi$. Moreover, P_g is composed of at most two directed paths. Then, at the connection between P_g and P_f in $P_5 = P_g + P_f$, the directions of two edges are same. By the birth order, once a directed path goes out from V_{small}, it never comes back inside V_{small}. Then P_5 satisfies the condition.

- When P_3 is composed of two directed paths:

Each of P_a, P_c, P_d and P_f is composed of a directed path. P_b is composed of two directed paths. Then, P_g is composed of at most two directed paths.

1. When P_g is composed of a directed path:
   The directions of edges are different either at the connection of P_d and P_g, or at the connection of P_g and P_f. Then, P_5 is an indirect distance of v_iv_j.
2. When P_g is composed of two directed path:
   Since the directions of edges are same at the connection of P_d and P_g, and P_g and P_f, P_5 is an indirect distance of v_iv_j.

Then, length(P_3) = length(P_4). □

In the following part of this chapter, we show PedigreeEnumerate which can output the solution of Pedigre Graph Enumerating Problem in $O(n^3)$ time as a directed graph in which the number of vertices is as same as the basic tournament. Before explaining the algorithm formally, we use examples in order to explain it intuitively.

Suppose that a distance matrix as shown in Fig. 4.7 (a) is given as the input of Pedigree Graph Enumerating Problem. The correspondig basic tournament is as shown in Fig. 4.7 (b). The algorithm should select two in-edges at each nodes. From Lemma 4.4, the selections can be done independently. For the simplicity of the explanation, the algorithm treats vertices in the ascending order, that is, $v_1, v_2, \ldots, v_n$.

From Lemma 4.3, the indegrees of v_1 and v_2 are zero. Moreover, the parents of v_3 are v_1 and v_2 (Fig. 4.7 (c)).

Next, the algorithm selects two edges which are terminated by v_4. In the basic tournament shown in Fig. 4.7 (b), the lengths of edges which are terminated by v_4 are 8, 3, and 5. From Lemma 4.1, if there exists a solution, (v_2, v_4) whose length is 3 must be included in G_{out}. Actually, as shown in Fig. 4.9 (a), (v_2, v_4) doesn't contradict the distance matrix.

Then, as shown in Fig. 4.9 (b), the algorithm investigates whether the solution contradicts the distance matrix or not when (v_3, v_4) is included by the solution. Although the distance of v_1v_4 is 6 in the pedigree graph shown in Fig. 4.9 (b), the correspoinding value in the distance matrix is 8. When an edge is added to the solution, there is a

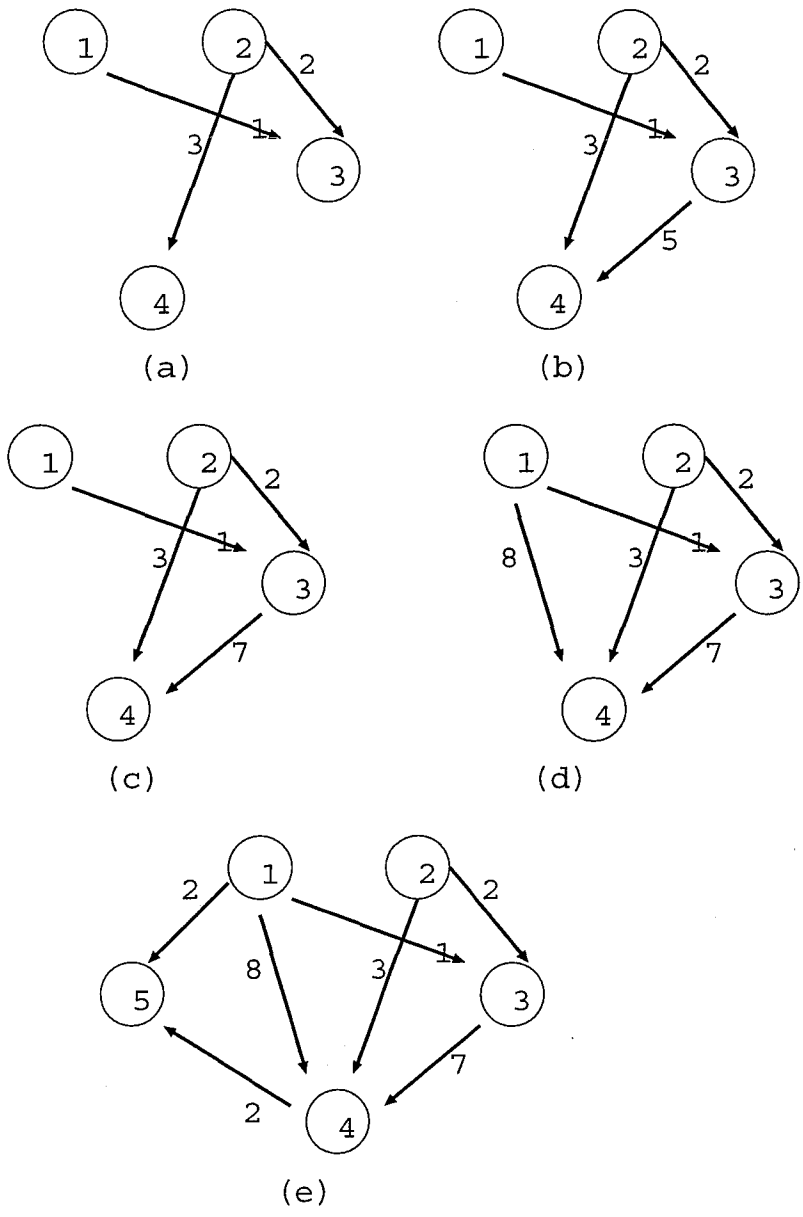


Figure 4.9: Selecting edges at v_4 and v_5

case that the distance of two vertices are shorter than the corresponding value of the distance matrix. In such a case, the algorithm can avoid the contradiction by increasing the length of the edge using the method shown in Lemma 4.5. In this example, the contradiction is eliminated by making the length of (v_3, v_4) 7 as shown in Fig. 4.9 (c). Thus, when the selected edges terminated by v_4 are (v_2, v_4) or (v_3, v_4), it is shown that the condition of the distance matrix is satisfied.

From Lemma 4.1, (v_2, v_4) should be included by the solution. There is a possibility that the condition of the distance matrix is satisfied when (v_1, v_4) is selected as the another edge terminated by v_4. In fact, there is no contradiction when (v_1, v_4) and (v_2, v_4) are selected as the two in-edges terminated by v_4. In other words, at the point of time shown in Fig. 4.9 (d), PedigreeEnumerate should keep three in-edges terminated by v_4. This means that the shortest edge (v_2, v_4) must be included by the solution and that both (v_1, v_4) and (v_3, v_4) can be the candidate for the other in-edge terminated by v_4.

Next, the algorithm selects edges terminated by v_5. The lengths of edges terminated by v_5 are 2, 5, 3 and 2. From Lemma 4.1, when there exists a solution, the shortest edges (v_1, v_5) and (v_4, v_5) should be included by the solution. Since the pedigree graph shown in Fig. 4.9 (e) which includes (v_1, v_5) and (v_4, v_5) satisfies the condition of the distance matrix, edges termiated by v_5 are (v_1, v_5) and (v_4, v_5).

Thus, PedigreeEnumerate outputs a directed graph shown in Fig. 4.9 (e) as the solution. As explained above, there is a possibility that an indegree of a vertex of the directed graph made by the algorithm is more than two. In such a case, solutions can be made by selecting the shortest edge and selecting the other edge arbitrary at each vertices. Details of PedigreeEnumerate are as follows:

algorithm PedigreeEnumerate

begin

1. Sort in-edges by lenghts in the ascending order at each vertices.
2. By the method of Lemma 4.3, find vertices whose indegrees are zero. Apply following procedures to vertices, denoted by $v_i (i = 1, \ldots, n)$, whose indegrees are not zero.

(a) From Lemma 4.2, length(e_1) $\neq$length(e_3).
when length(e_1)=length(e_2), eliminate all e_e ($e \geq 3$) and finish the process for v_i. Since there is a possibility that the distance matrix does'nt have a solution, investigate whether there is a contradiction or not when edges terminated by v_i are e_1 and e_2. When length(e_1)$\neq$ length(e_2), apply following procedures. Caluculate the shortest distances which use e_{i_1} between v_i and the other vertices. Aplly following procedures to e_{i_j} ($j \geq 2$).

i. Calculate the shortest distances which use e_{i_j} between v_i and the other vertices.

ii. Calculate the shortest distances between v_i and the other verrtices by comparing the case which uses e_{i_1} and the case which uses e_{i_j}. If the shortest distance is longer than the corresponding value of the distance matrix, since it is a contradiction, eliminate e_{i_j}. On the other hand, if the shortest distance is shorter than the corresponding value of the distance matrix, increase the length of the edge by the method of Lemma 4.5. If the contradiction still exists, eliminate e_{i_j}.

end

Suppose that edges terminated by v_i in G_{out} are e_{i_1} and e_{i_j}. Let $(d_{given_1}, d_{given_2}, \ldots, d_{given_{i-1}})$ be the distances which v_i should satisfy for $v_1, v_2, \ldots, v_{i-1}$. Let $(d_{1_1}, d_{1_2}, \ldots, d_{1_{i-1}})$ be the distances which use e_{i_1} between v_i and $v_1, v_2, \ldots, v_{i-1}$. Moreover, let $(d_{j_1}, d_{j_2}, \ldots, d_{j_{i-1}})$ be the distances which use e_{i_j} between v_i and $v_1, v_2, \ldots, v_{i-1}$.

Then, the algorithm compares d_{1_k}, d_{j_k} and $d_{given_k}(1 \leq k \leq i-1)$. In other words, the algorithm compares distances which use e_{i_1} or e_{i_j} and the corresponding values of the distance matrix. If d_{j_k} is longer than d_{given_k}, the algorithm memorizes "+". If d_{j_k} is as long as d_{given_k}, the algorithm memorizes "=". If d_{j_k} is shorter than d_{given_k}, the algorithm memorizes "−". (Fig. 4.10 shows an example.)

From Lemma 4.1, there are no "−" in $(d_{1_1}, d_{1_2}, \ldots, d_{1_{i-1}})$. If $(d_{1_1}, d_{1_2}, \ldots, d_{1_{i-1}})$ are all "=", since edges except for e_{i_1} are redundant, the algorithm should eliminate all edges terminated by v_i in G_{out} except for e_{i_1}. Then, we discuss only the case in which $(d_{1_1}, d_{1_2}, \ldots, d_{1_{i-1}})$ are composed of "=" and "+". $(d_{j_1}, d_{j_2}, \ldots, d_{j_{i-1}})$ are composed of "=", "+" and "−".

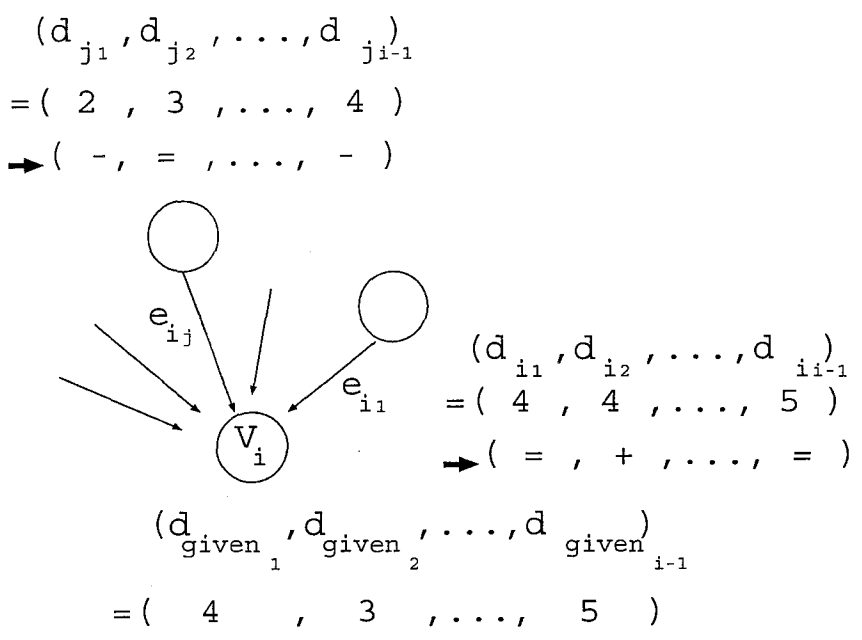


Figure 4.10: Comparing distances

Lemma 4.5 *When $d_{j_k}(1 \leq k \leq i-1)$ includes "−", e_{i_j} is included by G_{out} only if the length of e_{i_j} is modified to length(e_{i_j})+max$\{d_{given_k} - d_{j_k} | 1 \leq k \leq i-1\}$.*

Proof.

If length(e_{i_j}) is shorter than the value shown above, $d_{j_k}(1 \leq k \leq i-1)$ still includes "−". Then, it contradicts the distance matrix when e_{i_j} is included by G_{out}. On the other hand, if length(e_{i_j}) is shorter than the value shown above, $d_{j_k}(1 \leq k \leq i-1)$ are all "+". $(d_{1_1}, d_{1_2}, \ldots, d_{1_{i-1}})$ includes at least one "+". Then, if e_{i_j} is included by G_{out}, the distance becomes too long and it is a contradiction. □

PedigreeEnumerate outputs a directed graph which means pedigree graphs which satisfy the given distance matrix and the birth order. Now, we explain the method of seeing the directed graph. Let e_{i_1} be the only one shortest edge terminated by v_i.

- When the indegree of v_i is zero:

  It means that the given distance matrix doesn't include any information about ancestors of v_i.

- When the indegree of v_i is one:

  It means that the given distance matrix doesn't include the information about ancestors of one parent of v_i, but ancestors of the other parent of v_i.

- When the indegree of v_i is two:

  If the lengths of two edges terminated by v_i are same, both parents of v_i are uniquely determined.

- When the indegree of v_i is more than two:

  G_{out} must include e_{i_1}. The second edge can be selected arbitrary.

Theorem 4.1 *When G_{in} is given, the set of G_{out} which satisfies G_{in} can be found in $O(n^3)$ time by PedigreeEnumerate.*

Proof.

The distances which use e_{i_1} (or e_{i_j}) between v_i and the other vertices can be calculated by adding length(e_{i_1}) (or length(e_{i_j})) to the distance tables owned by adjacent vertices of v_i. □

In Theorem 4.1, we showed that PedigreeEnumerate can solve Pedigree Graph Enumerating Problem in $O(n^3)$ time. The inputs of this problem are a distance matrix and a birth order. The output of this problem is a set of pedigree graphs which satisfy the condition. On the other hand, Pedigree Graph Validating Problem is as follows: The inputs of the problem are a distance matrix, a birth order and a pedigree graph. The output is whether there is a contradiction or not.

Pedigree Graph Validating Problem

- Input: A distance matrix, a birth order, a pedigree graph.
- Question: Does the pedigree graph satisfy the given distance matrix and the birth order?

This problem can be solved in $O(n^2)$ time by applying the PedigreeEnumerate.

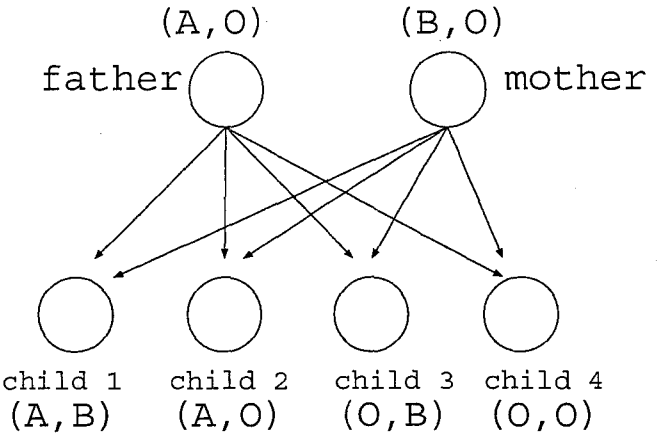


Figure 4.11: Genotypes of the blood type gene in the ABO blood system

Theorem 4.2 *When a distance matrix, a pedigree graph and a birth order are given, we can check whether there is a contradiction or not in $O(n^2)$ time.*

Proof.

PedigreeEnumerate investigates $O(n)$ edges which are terminated by each vertices of G_{in} whether they satisfy the distance matrix or not. Since the pedigree graph is given in Pedigree Graph Validating Problem, the numbers of edges which should be investigated at each vertices are two. Then, the algorithm can investigate whether there is a contradiction or not in $O(n^2)$ time. □

4.4 Inferring Pedigrees from Similarities of DNA

In the previous section, we dealt with problems to infer pedigrees from direct distances and indirect distances. However, there is a possibility that the number of output pedigrees is exponential of inputs and problems are made by theoretical interests. Then, the future works are as follows: (i) We want to determine the output pedigree graph uniquely. (ii) Problems should be applied to the practical use. In this section, we introduce some results for the future works.

The human genome has 20000~30000 genes. Each gene is composed of two alleles.

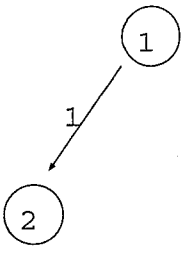


Figure 4.12: Node 1 is a parent of node 2 (the first ancestor).

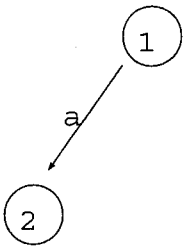


Figure 4.13: Node 1 is the a'th ancestor of node 2.

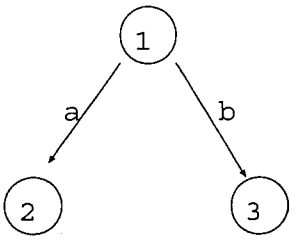


Figure 4.14: The similarity of the indirect distance is $1/2^{a+b}$.

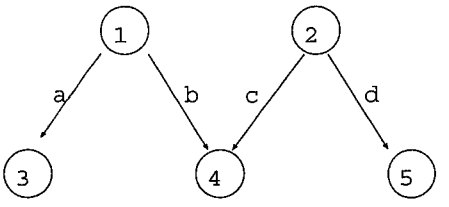


Figure 4.15: The case in which there are no similarity

Pairs of two genes are called *genotypes*. The total physical appearance that is caused by the genotype is called the *phenotype*.

For example, there are 3 alleles of the blood type gene: A, B, and O. In Fig. 4.11, the genotype of the father is (A, O). Since A is dominant and O is recessive, the phenotype of the father is A. Similarly, the genotype of the mother in Fig. 4.11 is (B, O). Since B is dominant and O is recessive, the phenotype of the mother is B. Children of the parents receive one gene from the father and the other gene from the mother. Then, genotypes of children can be (A, B), (A, O), (O, B), and (O, O) in this case. Each genotype occurs with the same probability from Mendel's law. Each allele of the parent is received by the child with probability 1/2. For example, in Fig. 4.12, an allele of node 1 is received by node 2 with probability 1/2. Numbers that are weighted to edges represent generations between nodes. Then, node 1 is a parent of node 2 in Fig. 4.12. If there is not another edge, an allele of node 1 is received by node 2 with probability $1/2^a$ in Fig. 4.13. Moreover, node 2 has a same allele that node 3 has with the probability $1/2^{a+b}$ in Fig. 4.14.

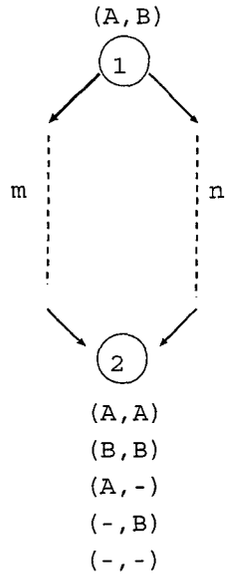


Figure 4.16: Genotypes inherited through 2 different paths

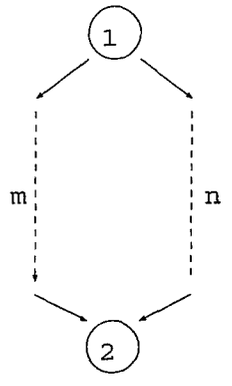


Figure 4.17: (m, n)-paths

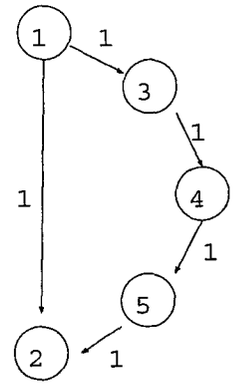


Figure 4.18: $(1, 4)$-paths

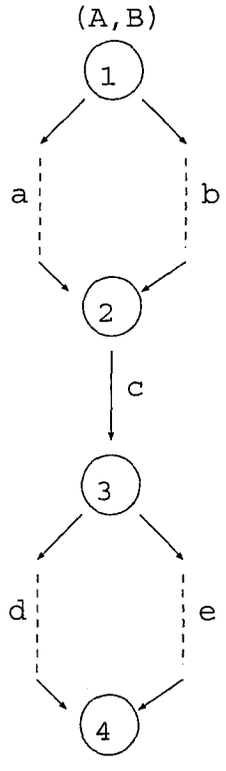


Figure 4.19: 4 different paths

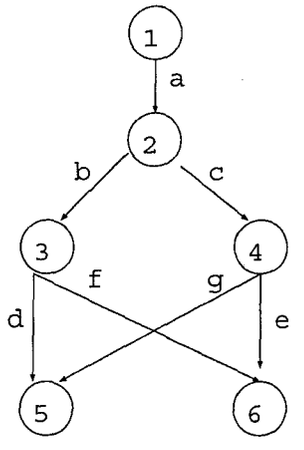


Figure 4.20: Indirect similarity

Although the ABO blood system has only 3 alleles, our inference method needs much more alleles. Then, we consider sets of genes or SNPs instead of using one gene. Although details are explained in the next paragraph, we can assume that Node 3 does not have same alleles that node 2 or node 5 have in Fig. 4.15.

In Fig. 4.16, the ancestor has an allele (A, B). Since there are 2 edge-disjoint paths, the descendant can have the following alleles: (A, A), (B, B), (A, –), (–, B), and (–, –). "–" means alleles that do not come from the ancestor. Moreover, there is a possibility that "–" is "A" or "O". Although our purpose is to infer a pedigree graph from similarities of DNA that are approximated by the ratio of same genotypes between two individuals, the number of alleles, that is 3 in this case, is too small. Then, we consider direct products of genes. For example, suppose that genes $g_i (i = 1, ..., k)$ have a_i alleles. Then, the direct product of genes $S = g_1 \times g_2 \times \cdots \times g_k$ has a genotype that has $a_1 \times a_2 \times \cdots \times a_k$ alleles. If k is large enough, we can assume that every node in which the indegree is zero has different genotypes, and does not have same alleles each other. Thus, the ratio of same genotypes between two nodes that are connected by only one a-length path is $1/2^a$.

However, pedigree graphs have cycles. If a pedigree graph has many cycles, it needs exponential times of inputs to validate whether the solution is feasible or not. Suppose that there are 2 node that are connected by 2 different paths as shown in Fig. 4.17. Suppose that the length of one path is m and the length of the other path is n. This pedigree graph is called (m, n)-*paths*. For example, the pedigree graph shown in Fig. 4.18 is $(1, 4)$-paths.

Since $m + n$ is very large in every (m, n)-paths included in a pedigree graph in general, it is quite rare that node 2 receive either (A, A) or (B, B) from node 1. Then, in Fig. 4.16, the ratio of same genotypes between node 1 and node 2 can be approximated by $1/2^m + 1/2^n$.

From the approximation, the *similarity between* v_i *and* v_j, that is denoted by $S(v_i, v_j) = DS(v_i, v_j) + IS(v_i, v_j)$, is defined as follows: When v_i and v_j are connected by p different direct paths and the length of each direct path is $l_k (k = 1, \ldots, p)$, the *direct similarity* between v_i and v_j, that is denoted by $DS(v_i, v_j)$, is $\sum_{k=1}^{p} 1/2^{l_k}$. For example, in Fig. 4.17, $DS(v_i, v_j) = 1/2^m + 1/2^n$. Moreover, in Fig. 4.19, $DS(v_1, v_4) = 1/2^{a+c+d} + 1/2^{a+c+e} + 1/2^{b+c+d} + 1/2^{b+c+e}$.

Although there is a possibility that the number of different paths is exponential of nodes, the direct similarity between 2 nodes can be calculated in polynomial times by using the topological order.

Next, we define *indirect similarity*. When v_i and v_j are connected by p different indirect paths and the length of each indirect path is $l_k (k = 1, \ldots, p)$, the indirect similarity between v_i and v_j, that is denoted by $IS(v_i, v_j)$, is $\sum_{k=1}^{p} 1/2^{l_k}$. For example, in Fig. 4.20, $IS(v_5, v_6) = 1/2^{d+f} + 1/2^{g+e} + 1/2^{d+b+c+e} + 1/2^{g+c+b+f}$.

Our goal is to infer the corresponding pedigree graph from given $S(v_i, v_j)$. The following lemma plays one of the most basic part of the inference.

Lemma 4.6 *If v_i and v_j have one of the following relationship, $S(v_i, v_j) \geq 1/2$: (i) v_i is a parent of v_j, (ii) v_j is a parent of v_i, and (iii) v_i and v_j are brothers.*

Proof.

When either (i) or (ii) is satisfied, $S(v_i, v_j) \geq DS(v_i, v_j) \geq 1/2$. When (iii) is satisfied, $S(v_i, v_j) \geq IS(v_i, v_j) \geq 1/2^2 + 1/2^2 = 1/2$. □

The inversion of Lemma 4.6 is not true. Suppose that a pedigree graph includes $(2, 2)$-paths as shown in Fig. 4.21. Although node 1 and node 4 do not satisfy any condition of Lemma 4.6, $S(v_1, v_4) = 1/2^2 + 1/2^2 = 1/2$.

However, if $(m + n)$ of each (m, n)-paths is large enough, there is a possibility that the inversion of Lemma 4.6 is satisfied. In the following lemma, we discuss about the case in which $m + n \geq 5$.

Lemma 4.7 *Suppose that $m + n \geq 5$ and $m, n \neq 1$ in every (m, n)-paths. v_i and v_j have one of the following relationships if and only if $IS(v_i, v_j) \geq 1/2$: (i) v_i is a parent of v_j, (ii) v_j is a parent of v_i, and (iii) v_i and v_j are brothers.*

Proof.

We can assume that v_i is an ancestor of v_j without loss of generality. Let v_k be a descendant of v_i and an ancestor of v_j. $IS(v_i, v_k)$ can be calculated by using the topological order as following: Suppose that v_{k_p} is only one parent of v_k. Then,

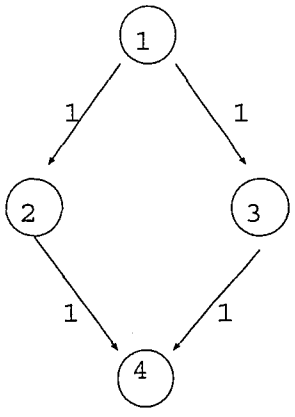


Figure 4.21: (2,2)-paths

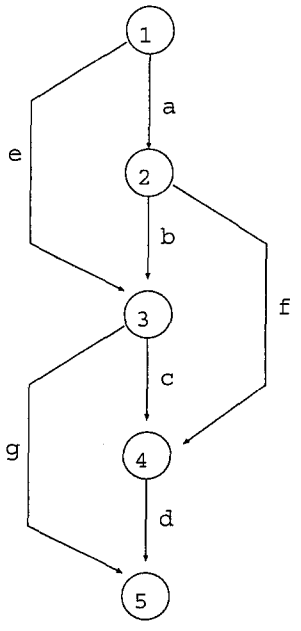


Figure 4.22: An example

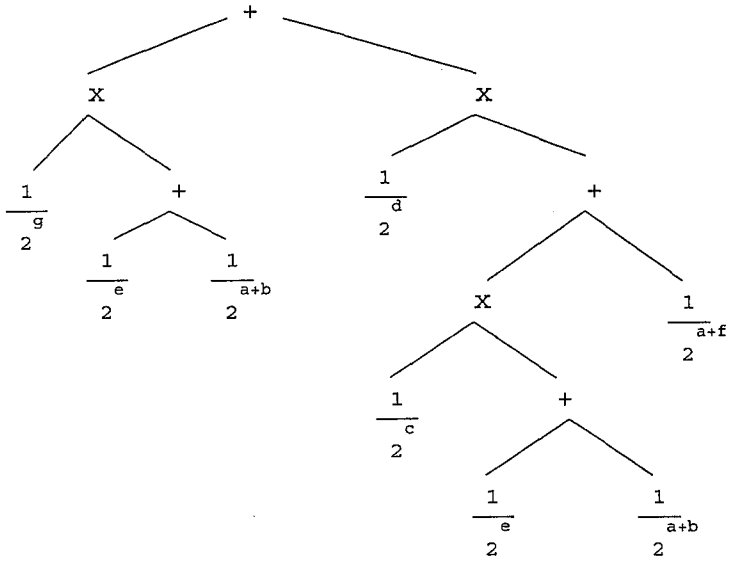


Figure 4.23: The corresponding binary tree

$IS(v_i, v_k) = \frac{IS(v_i, v_{k_p})}{2}$. Similarly, let v_{k_f} and v_{k_m} be parents of v_k. Then, $IS(v_i, v_k) = \frac{IS(v_i, v_f)}{2} + \frac{IS(v_i, v_m)}{2}$. Thus, (v_i, v_j) can be calculated. For example, in Fig. 4.22,

- $IS(v_1, v_2) = \frac{1}{2^a}$,
- $IS(v_1, v_3) = \frac{1}{2^{a+b}} + \frac{1}{2^e}$,
- $IS(v_1, v_4) = \frac{1}{2^c}(\frac{1}{2^{a+b}} + \frac{1}{2^e}) + \frac{1}{2^{a+f}}$, and
- $IS(v_1, v_5) = \frac{1}{2^g}(\frac{1}{2^{a+b}} + \frac{1}{2^e}) + \frac{1}{2^d}\{\frac{1}{2^c}(\frac{1}{2^{a+b}} + \frac{1}{2^e}) + \frac{1}{2^{a+f}}\}$.

In order to prove the lemma, we construct binary trees that correspond these mathematical formulas. For example, the binary tree shown in Fig. 4.23 corresponds to the formula of $IS(v_1, v_5)$. The calculations are done in a topological order from bottom up in the binary tree. Since $m + n \geq 5$ and $m, n \neq 1$, calculations between leaf nodes are $\frac{1}{2^m} + \frac{1}{2^n} < \frac{1}{2}$.

Since indegrees are at most 2 in a pedigree graph, neighbor nodes of "+" are never "+". Then, if a child node of "+" is an internal node, the result of the calculation at the child node is less than 1/4. Since $m, n \neq 1$, children of "+" are at most 1/4. Then, calculation results of "+" are less than 1/2. Since every "×" is done after "+", calculation results of "×" are less than 1/4. Thus, since every calculation result is less than 1/2, the lemma is satisfied. □

Let G_{IS} be a graph and $v_i, v_j \in V(G_{IS})$. If $IS(v_i, v_j) \geq 1/2$, G_{IS} has an edge between v_i and v_j. From Lemma 4.7, we can infer a pedigree graph by finding pseudo cliques that are defined as follows: Let $V_c \subseteq V_{IS}$. Let $G_{IS}[V_c] = (V_c, E_c)$ be an induced subgraph of G_{IS} by V_c. Let $K^{|V_c|} = (V_k, E_k)$ be a complete graph that has $|V_c|$ vertices. When $(V_k, E_k) = (V_c, E_c + e)$, $G_{IS}[V_c]$ is called a *pseudo clique* ($e = v_f v_m$ and $v_f, v_m \in V_c$). For example, the graph shown in Fig. 4.24 is a pseudo clique. (v_f, v_m) is either (v_1, v_2) or (v_2, v_1).

Moreover, a pseudo clique of G_{IS} corresponds to a nuclear family. For example, a pseudo clique as shown in Fig. 4.24 corresponds to a nuclear family as shown in Fig. 4.11. v_f is the father and v_m is the mother.

However, for practical use, we cannot get $IS(v_i, v_j)$ but $S(v_i, v_j)$. Then, we want to use $S(v_i, v_j)$ instead of $IS(v_i, v_j)$ for Lemma 4.7. However, there are counterexamples.

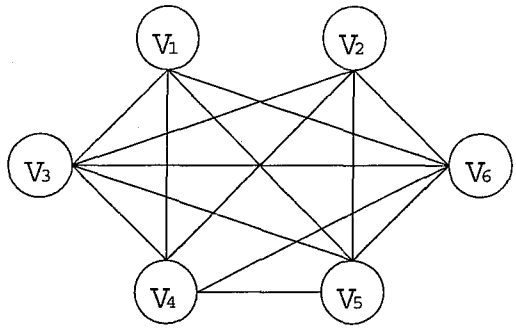


Figure 4.24: A pseudo clique

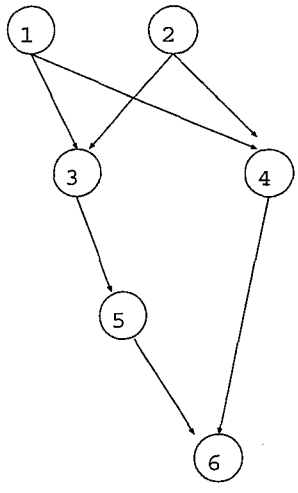


Figure 4.25: A counterexample in which $S(v_3, v_6) = 1/2$

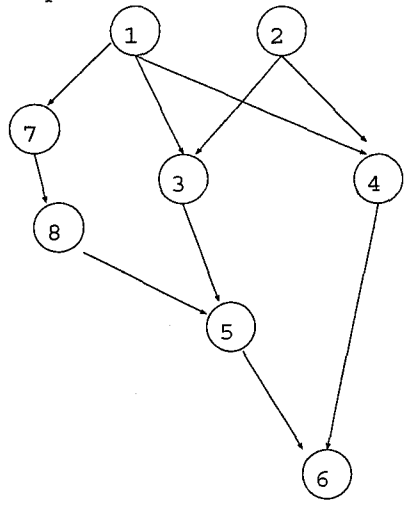


Figure 4.26: A counterexample in which $S(v_3, v_6) = 17/32$

Although node 3 and node 6 do not have any relationship of Lemma 4.7 in Fig. 4.25, $S(v_3, v_6) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$. Moreover, $S(v_3, v_6) = \frac{1}{4} + \frac{9}{32} = \frac{17}{32}$

4.5 Concluding Remarks

In this chapter, we formulated problems to infer pedigrees from genetic distances and a birth order as Pedigree Graph Enumerating Problem. We also formulated problems to investigate whether there is a contradiction or not among given genetic distances, a

birth order and a pedigree graph as Pedigree Graph Validating Problem.

We showed $O(n^3)$ time algorithms for Pedigree Graph Enumerating Problem and $O(n^2)$ time algorithms for Pedigree Graph Validating Problem.

Chapter 5

Conclusion

In this thesis, algorithms that infer structures in bioinformatics from graph algorithmic approaches.

In Chapter 2, for the problem of fixing the probe order of a given PQ-tree, we presented polynomial time algorithms that minimize the number of additional fragments. We solved not only the problem of fixing probes in a given order, but also the problem to how to determine the best order of the probes. By using an idea of "edges", we showed that the minimum number of additional fragments are $\lceil \frac{e}{2} \rceil$ or $\lceil \frac{e}{2} \rceil + 1$, where e is the number of edges. For practical use, there is the possibility that H-matrices include errors. Then, a future work of ours will be to consider STS-based mapping with errors.

In Chapter 3, we define another measure of imperfectness and construct polynomial time algorithms that calculate minimum fragment sets. The second measure of imperfectness is the total length of the additional fragments that are needed to uniquely fix the probe order. We also mention cases that additional fragments are restricted to sub-fragments of existed fragments, and that the lengths of additional fragments are limited to 2.

In Chapter 4, we formulated problems to infer pedigrees from genetic distances and a birth order as Pedigree Graph Enumerating Problem. We also formulated problems to investigate whether there is a contradiction or not among given genetic distances, a birth order and a pedigree graph as Pedigree Graph Validating Problem. We showed

$O(n^3)$ time algorithms for Pedigree Graph Enumerating Problem and $O(n^2)$ time algorithms for Pedigree Graph Validating Problem.

Throughout this thesis, we discussed about the algorithms that infer structures in bioinformatics from graph algorithmic approaches. We believe that our inferring algorithms would be of some help for revealing the biological meanings of genomes. And, we also hope that every post-genome research in the world would be supported by efficient algorithms that are constructed with robust theory.

Bibliography

[1] L. Aceto, J. A. Hansen, A. Ingolfsdottir, J. Johnsen, and J. Knudsen, The complexity of checking consistency of pedigree information and related problems, Journal of Computer Science and Technology, 19(1): pp. 42–59, 2004.

[2] A. V. Aho, J. E. Hopcraft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions, SIAM Journal on Computing, 10(3): pp. 405–421, 1981.

[4] F. Alizadeh, R. M. Karp, L. A. Newberg, and D. K. Weisser, Physical mapping of chromosomes, a combinatorial problem in molecular biology. Algorithmica, 13(1/2): pp. 52–76, 1995.

[5] F. Alizadeh, R. M. Karp, D. K. Weisser, and G. Zweig, Physical mapping of chromosomes using unique probes, Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 489–500, 1994.

[6] I. Althofer, On optimal realizations of finite metric spaces by graphs, Discrete & Computational Geometry, (3): pp. 103–122, 1988.

[7] F. S. Annexstein and R. P. Swaminathan, On testing consecutive-ones property in parallel, Proceedings of the Seventh annual ACM symposium on Parallel Algorithms and Architectures (SPAA), pp. 234–243, 1995.

[8] R. Arratia, E. S. Lander, S. Tavare, and M. S. Waterman, Genomic mapping by anchoring random clones: a mathematical analysis, Genomics, (11): pp. 806–827, 1991.

[9] R. Beigel, N. Alon , S. Kasif , M. S. Apaydin, and L. Fortnow, An optimal procedure for gap closing in whole genome shotgun sequencing, Research in Computational Molecular Biology (RECOMB), pp. 22–30, 2001.

[10] A. Ben-Dor and B. Chor, On constructing radiation hybrid maps (extended abstract), Research in Computational Molecular Biology (RECOMB), pp. 17–26, 1997.

[11] V. Berry and O. Gascuel, Inferring evolutionary trees with strong combinatorial evidence, Theoretical Computer Science, 240(2): pp. 271–298, 2000.

[12] T. Beyer and S. M. Hedetniemi, Constant time generation of rooted trees. SIAM Journal on Computing, 9(4), pp. 706–712, 1980.

[13] K. S. Booth and G. S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, Journal of Computer and System Sciences, (13): pp. 335–379, 1976.

[14] T. A. Brown, Gene cloning, Chapman and Hall, second edition, 1990.

[15] D. Bryant and V. Moulton, Neighbor Net, An agglomerative method for the construction of planar phylogenetic networks, Algorithms in Bioinformatics (WABI), pp. 375–391, 2002.

[16] F. Y. L. Chin, Q. Zhang, and H. Shen, k-recombination haplotype inference in pedigrees, International Conference on Computational Science (ICCS), (2): pp. 985–993, 2005.

[17] B. Chor and M. Sudan, A geometric approach to betweenness, European Symposium on Algorithms (ESA), pp. 227–237, 1995.

[18] C. Choy, J. Jansson, K. Sadakane, and W. K. Sung. Computing the maximum agreement of phylogenetic networks, Electronic Notes in Theoretical Computer Science (ENTCS), (91): pp. 134–147, 2004.

[19] T. Christof, M. Jnger, J. Kececioglu, P. Mutzel, and G. Reinelt, A branch-and-cut approach to physical mapping with end-probes, Research in Computational Molecular Biology (RECOMB), pp. 84–92, 1997.

[20] T. Christof and J. Kececioglu, Computing physical maps of chromosomes with nonoverlapping probes by branch-and-cut, Research in Computational Molecular Biology (RECOMB), pp. 115–123, 1999.

[21] D. Cohen, I. Chumakov, and J. Weissenbach, A first-generation physical map of the human genome, Nature, (366): pp. 698–701, 1993.

[22] K. Doi, J. Li, and T. Jiang, Minimum recombiant haplotype configuration on tree pedigrees, Algorithms in Bioinformatics (WABI), pp. 339–353, 2003.

[23] D. P. Fasulo, T. Jiang, R. M. Karp, R. Settergren, and E. C. Thayer, An algorithmic approach to multiple complete digest mapping, Research in Computational Molecular Biology (RECOMB), pp. 118–127, 1997.

[24] J. Felsenstein, Numerical methods for inferring evolutionary trees, Quarterly Review on Biology, 57(4): pp. 379–404, 1982.

[25] J. Felsenstein, PHYLIP: Phylogeny inference package (version 3.2), Cladistics 5: pp. 164–166, 1989.

[26] D. F. Feng and R. F. Doolittle, Progressive alignment of amino acid sequences and construction of phylogenetic trees from them, Methods Enzymol, (266): pp. 368–382, 1996.

[27] W. M. Fitch, Toward defining the course of evolution: Minimal change for a specific tree topology, Systematic Zoology, (20): pp. 406–441, 1971.

[28] W. M. Fitch and E. Margoliash, Construction of phylogenetic trees, Science, (155): pp. 279–284, 1987.

[29] M. R. Garey and D. S. Johnson, Computer and intractability, A guide to the theory of NP-completeness, 1979.

[30] D. S. Greenberg and S. Istrail, Physical mapping by STS hybridization, algorithmic strategies and the challenge of software evaluation, Journal of Computational Biology, 2(2): pp. 219–273, 1995.

[31] D. Gusfield, Efficient algorithms for inferring evolutionary trees, Networks, (21): pp. 19–28, 1991.

[32] D. Gusfield, S. Eddhu, and C. Langley, Efficient reconstruction of phylogenetic networks with constrained recombination, IEEE Computer Society Bioinformatics Conference (CSB), pp. 363–374, 2003.

[33] M. C. Golumbic, H. Kaplan, and R. Shamir, On the complexity of DNA physical mapping, Advances in Applied Mathematics, (15): pp. 251–261, 1994.

[34] M. T. Hajiaghayi and Y. Ganjali, A note on consecutive ones submatrix problem, Information processing letters, (83): pp. 163–166, 2002.

[35] S. L. Hakimi and S. S. Yau, Distance matrix of a graph and its realizability, Quarterly Journal of Applied Mathematics, (22): pp. 305–317, 1965.

[36] L. S. Heath and S. V. Pemmaraju, Recognizing leveled-planar dags in linear time, Graph Drawing, pp. 300–311, 1995.

[37] S. Heber, J. Hoheisel, and M. Vingron, Application of bootstrap techniques to physical mapping, Genomics, (69): pp. 235–241, 2000.

[38] W. Hsu, On physical mapping algorithms - an error-tolerant test for the consecutive ones property, Computing and Combinatorics, pp. 242–250, 1997.

[39] D. H. Huson, T. Dezulian, T. Klopper, and M. A. Steel, Phylogenetic supernetworks from partial trees, Algorithms in Bioinformatics (WABI), pp. 388–399, 2004.

[40] M. Jain and E. W. Myers, Algorithms for computing and integrating physical maps using unique probes, Research in Computational Molecular Biology (RECOMB), pp. 151–161, 1997.

[41] M. Jain and G. Myers, A note on scoring clones given a probe ordering, Journal of Computational Biology, 2(1): pp. 33–38, 1995.

[42] J. Jansson and W. K. Sung, Inferring a level-1 phylogenetic network from a dense set of rooted triplets, Computing and Combinatorics (COCOON), pp. 462–472, 2004.

[43] J. Jansson and W. K. Sung, The maximum agreement of two nested phylogenetic networks, International Symposium on Algorithms and Computation (ISAAC), pp. 581–593, 2004.

[44] T. Jiang and R. M. Karp, Mapping clones with a given ordering or interleaving, Symposium on Discrete Algorithms (SODA), pp. 400–409, 1997.

[45] M. Junger, S. Leipert, and P. Mutzel, Level planarity testing in linear time, Graph Drawing, pp. 224–237, 1998.

[46] R. M. Karp and R. Shamir, Algorithms for optical mapping, Research in Computational Molecular Biology (RECOMB), pp. 117–124, 1998.

[47] J. Kececioglu, S. Shete, and J. Arnold, Reconstructing distances in physical maps of chromosomes with nonoverlapping probes, Research in Computational Molecular Biology (RECOMB), pp. 183–192, 2000.

[48] E. Knill, Lower bounds for identifying subset members with subset queries, Symposium on Discrete Algorithms (SODA), pp. 369–377, 1995.

[49] E. S. Lander and M. S. Waterman, Genomic mapping by finger printing random clones; a mathematical analysis, Genomics, (2): pp. 231–239, 1988.

[50] J. K. Lee, V. Dancik, and M. S. Waterman, Estimation for restriction sites observed by optical mapping using reversible-jump Markov chain Monte Carlo, Research in Computational Molecular Biology (RECOMB), pp. 147–152, 1998.

[51] J. Li and T. Jiang, An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming, Research in Computational Molecular Biology (RECOMB), pp. 20–29, 2004.

[52] J. Li and T. Jiang, Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming, Journal of Computational Biology, 12(6): pp. 719–739, 2005.

[53] J. Li and T. Jiang, Efficient rule-based haplotyping algorithms for pedigree data, Proceedings of the 7th annual international conference on Computational molecular biology, pp. 197–206, 2003.

[54] Z. Li and S. Nakano, Efficient generation of plane triangulations without repetitions, International Colloquium on Automata, Languages and Programming (ICALP), pp. 433–443, 2001.

[55] P. Little, Mapping the way ahead, Nature, (359): pp. 367–368, 1992.

[56] W. Lu and W. Hsu, A clustering algorithm for interval graph test on noisy data, Workshop on Experimental and Efficient Algorithms (WEA), pp. 195–208, 2003.

[57] W. Lu and W. Hsu, A test for the consecutive ones property on noisy data - Application to physical mapping and sequence assembly, Journal of Computational Biology, 10(5): pp. 709–735, 2003.

[58] V. Makarenkov and P. Legendre, From a phylogenetic tree to a reticulated network, Journal of Computational Biology, 11(1): 195–212, 2004.

[59] E. Masood, Gene warrior, New Scientist, Magazine issue 2247, July 2000.

[60] G. Mayraz and R. Shamir, Construction of physical maps from oligonucleotide fingerprints data, Research in Computational Molecular Biology (RECOMB), pp. 268–277, 1999.

[61] R. Mott, A. Grigoriev, E. Maier, J. Hoheisel, and H. Lehrach, Algorithms and software tools for ordering clone libraries: application to the mapping of the genome of Schizosaccharomyces pombe, Nucleic Acids Research, 21(8): pp. 1965–1974, 1993.

[62] S. Muthukrishnan and L. Parida, Towards constructing physical maps by optical mapping (extended abstract): an effective, simple, combinatorial approach, Research in Computational Molecular Biology (RECOMB), pp. 209–219, 1997.

[63] G. Myers, Whole-genome DNA-sequencing, Computing in Science and Engineering, 1(3): pp. 33–43, 1999.

[64] S. Nakano, Efficient generation of plane trees, Information Processing Letters, 84(3): pp. 167–172, 2002.

[65] S. Nakano and T. Uno, Constant time generation of trees with specified diameter, Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp 33–45, 2004.

[66] N. Saitou and M. Nei, The neighbor-joining method, a new method for reconstructing phylogenetic trees, Molecular Biology Evolution, (4): pp. 406–425, 1987.

[67] S. Skiena, Implementing discrete mathematics, Combinatorics and Graph Theory with Mathematica, Addison-Wesley, pp. 175, 1990.

[68] D. K. Slonim, L. Kruglyak, L. Stein, and E. S. Lander, Building human genome maps with radiation hybrids, Journal of Computational Biology, 4(4): pp. 487–504, 1997.

[69] J. Srba, On counting the number of consistent genotype assignments for pedigrees, Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 470–482, 2005.

[70] K. Strimmer and A. von Haeseler, Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies, Molecular Biology and Evolution, 13(7): pp. 964–969, 1996.

[71] H. Thiele and P. J. Nurnberg, HaploPainter: a tool for drawing pedigrees with complex haplotypes, Bioinformatics, 21(8): pp. 1730–1732, 2005.

[72] A. Thomas, GMCheck: Bayesian error checking for pedigree genotypes and phenotypes, Bioinformatics, 21(14): pp. 3187–3188, 2005.

[73] D. Torney, Mapping using unique sequences, Journal of molecular biology, (217): pp. 259–264, 1991.

[74] Y. Wang, R. A. Prade, J. Griffith, W. E. Timberlake, and J. Arnold, A fast random cost algorithm for physical mapping, Proceedings of the National Academy of Sciences, (91): pp. 11094–11098, 1994.

[75] L. Wang, K. Zhang, and L. Zhang, Perfect phylogenetic networks with recombination, Journal of Computational Biology, 8(1): pp. 69–78, 2001.

[76] E. A. Wernert and J. Lakshmipathy, PViN: A scalable and flexible system for visualizing pedigree databases, Selected Areas in Cryptography (SAC), pp. 115–122, 2005.

[77] D. B. Wilson, D. S. Greenberg, and C. A. Phillips, Beyond islands (extended abstract), Runs in clone-probe matrices, Research in Computational Molecular Biology (RECOMB), pp. 320–329, 1997.

[78] M. Xiong, H. J. Chen, R. A. Prade, Y. Wang, J. Griffith, W.E. Timberlake, and J. Arnold, On the consistency of a physical mapping method to reconstruct a chromosome in vitro, Genetics, (142): pp. 267–284, 1996.

[79] Q. Zhang, F. Y. L. Chin, and H. Shen, Minimum parent-offspring recombination haplotype inference in pedigrees, Transactions on Computational Systems Biology, (2): pp. 100–112, 2005.

[80] M. Q. Zhang and T. G. Marr, Genome mapping by nonrandom anchoring: a discrete theoretical analysis, Proceedings of the National Academy of Sciences, (90): pp. 600–604, 1993.

Publications

- Hiro Ito, Kazuo Iwama, and Takeyuki Tamura, "Efficient Methods for Determining DNA Probe Orders", IEICE Transactions, Vol. E88-A, No. 5, 2006. (to appear)
- Hiro Ito, Kazuo Iwama, and Takeyuki Tamura, "Imperfectness of Data for STS-Based Physical Mapping", Exploring New Frontiers of Theoretical Informatics (Proceedings of TCS2004), J.- J. Levy, E. W. Mayr, and J. C. Mitchell (Eds.), KLUWER, 2004, pp. 279–292, IFIP World Computer Congress, 2004/08/22–27, Toulouse, France.
- Takeyuki Tamura, Daisuke Tsuchida, Hiro Ito, and Kazuo Iwama, "Minimum Fragments for Deciding Probe Sequences for DNA Strands", Information Technology Letters, Vol.1, pp5-6, 2002. (in Japanese)

Technical Reports

- Takeyuki Tamura, Hiro Ito, and Kazuo Iwama, "Inferring Pedigrees from Genetic Distances", IEICE/COMP, Vol.104, No.642, pp33-39, January 28, 2005. (in Japanese)
- Takeyuki Tamura, Hiro Ito, and Kazuo Iwama, "Minimum Sets of Fragments for Fixing Probe Sequences for DNA Strands", IEICE/COMP, Vol.102, No.733, pp23-30, March 18, 2003. (in Japanese)